



User Manual

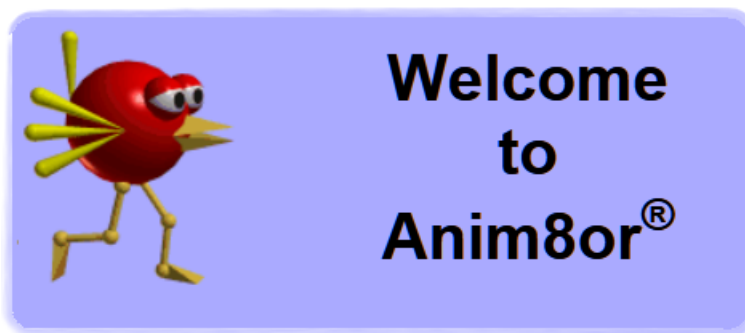
Build 1.01.1318

Copyright 2018 R. Steven Glanville

Anim8or v1.0

User Manual

Current development release is **v1.01.1318**, January 25, 2018



Like Anim8or, this manual is a work in progress. While it is intended to be a manual for Anim8or v1.0 it has been worked to be backwards compatible to Anim8or v0.95c and forwards compatible to Anim8or v1.01.1318. Obviously the older versions will be missing the newer functions and while Anim8or is in a state of development there may be changes to commands, functions, and buttons. Sorry about any redundancies or errors!

On the cover:

anim8or-splashscreen-thecolclough.png

Copyright 2017 thecolclough

Anim8or® is a registered trademark of R. Steven Glanville.

Copyright 2018 R. Steven Glanville



Contents

1. Introduction	... 18
•What's New in V1.0	19
•User Interface	
•Modeling	
•Animation	
•Features	
•System Requirements	21
•Windows	
•Linux	
•Installation Instructions	21
•News on update	
•News on development release	
•Compatibility	22
•File Formats	22
•Load and Import	
•Save and Export	
•Scripting Language Specification	22
•Anim8or Tutorials	23
•Anim8or Support	23
 2. Basics	 ... 24
•Anim8ors' New User Interface	24
•Menu and Toolbars	24
•Menu	
◦Modes and their menus	

•Top Toolbar	
•Side Toolbar	
•Common Button Meanings	
•Viewports	29
•Mouse Usage	30
•Keyboard Shortcuts	30
•Undo and Redo	30
•Tool Tips	30
•Project Description	30
•Arc Rotate	31
•Editing Widgets	32
•Grid Control	33
•Snap-to-Grid	
•Material Editor	33
•Anim8or Object Libraries	34
•Visual Quality	34
•OpenGL Workspace	
•File Output	
•User Attributes	35
•Saving .jpg, .png and .bmp File	36
•Printing	36
•Auto Save	36
•Configuration	36
•User Interface Configuration	37
•CAD Notations	38
•Alignment Ruler	
•Layers	39

3. Object Editor - Basics and Object/Edit Mode	... 41
•Object/Edit Tools	42
•Basic Objects	43
•Meshes vs. Parameteric Shapes	44
•Object Materials	45
•Plug-in Shapes	45
•Splines	45
•True Type Fonts	46
•Filling	46
•Extrusion	47
•Lathing	48
•Modifiers	49
•Mirror Image	49
•Smoothing	49
•Subdivision Objects	50
•Morph Targets	50
•Continuously Mirrored Meshes	51
•Reference Image	52

4. Object Editor - Object/Point Mode	... 53
•Points / Edges / Faces	53
•Normals	
•Object/Point Operations	54
•Point Editing	54
•Edge Editing	54
•Forum Note: Edge-Point Joint selection	

•Face Editing	56
•Applying Multiple Materials	56
•A Note on Selecting Faces	
•Adding Points and Edges	57
•Adding Faces	57
•Connecting Meshes	58
•Merging Points	59
•Connecting Meshes (2) by Merging Points	
•Connecting Meshes (3) by Merging Points	
•Point and Line Parameters	59
•Face Extrusion and Manipulating Tools	60
•Bevel	
•Inset	
•Shell	
•Bump	
•Cut Faces	
•Slide	
•Edge Extrude	
•Topographical Knife	63
•Split Faces	
•Add Edges	
•Dissolve	
•Move Edges and Points	
•Move by Normal	
•Face Editing Commands	64
•Bridge	
•Flatten	
•Merge Faces	

•Misc. Commands	66
•Selection Commands	
◦ <i>Quad Loop Select</i>	
◦ <i>Quad Ring Select</i>	
•Editing Commands	
◦ <i>Connect Edges</i>	
◦ <i>Cut Edges</i>	
◦ <i>Detach Faces</i>	
◦ <i>Subdivide Faces</i>	
◦ <i>Loop Cut</i>	
◦ <i>Spin Quads</i>	
◦ <i>Invert Selection</i>	
◦ <i>Grow Face Selection</i>	
◦ <i>Select Connected</i>	
◦ <i>Select by Material</i>	
◦ <i>Select Malformed Edges</i>	
◦ <i>Select Orphans</i>	
•Fast Select	67
5. Figure Editor	... 70
•Figure Basics	71
•Edit Operations	71
•Visibility	
•Building a Skeleton	74
•Flexible Joints	75
•Adding Body Parts	76
•Skinning	78

- Influence Volumes

- Skinning Weights

6. Sequence Editor ... **80**

- Time Track 80

- Scrubber Bar 81

- Sequence Basics 82

- Edit Operations 82

- Visibility Options 82

- Animate Button 83

- What is a Key?

- Making a Key Pose 83

- Using the Trackball

- Trackball Rotation for Bones

- Click-dragging on Bones

- Inverse Kinematics 84

- Forum Notes: Updated IK tool

- Editing Key Frames 87

- A Circle Has 720 Degrees 88

- Ghost Views 88

- Making a Sequence 89

- Exporting and Importing Sequences 89

7. Scene Editor ... **90**

- Scene Parameters 91

- Name

- Menu Folder

•Frames	
•Environment	
•Global Lighting	
•Movie Image	
•Elements of a Scene	93
•Adding Objects	93
•Adding Figures	93
•The Camera	95
•FOV	
•Lights	96
Infinite (or directional)	
•Local	
•Spotlights	
•Targets	97
•Object Properties Dialog	97
•Name	
•Object	
•Location (Position)	
•Parent	
•Orientation	
•Enable Roll	
•Visible	
•Shadows	
•Editing Properties	
•Other Elements	
•Light Properties	101
•Advanced Light Properties	103

• Shadows	103
•Volume Shadows	
•Ray Traced Shadows	
• Environment Settings	105
• Advanced Environment Editor	106
•Z Limits	
• The Time Track	107
• Previewing and Saving Images	108
 8. Scene Animation	 ... 109
• Animation with Key Frames	109
• Animating with Inverse Kinematics	109
•Forum Notes: Difference between Lock and Hold	
• Inverse Kinematic Walk Cycle	111
• Animation Figures with Sequences	111
• Animating with Expressions	112
• Controllers	112
•Kinds of Controllers	
◦ <i>Float</i>	
◦ <i>Point3</i>	
◦ <i>Orientation</i>	
◦ <i>Boolean</i>	
•Editing Keys	
•Editing Expressions	
•Graph Editor	
•Editing Segments	

9. Materials **... 117**

•Material Editor	117
•Name	
•Ambient	
•Diffuse	
•Specular	
•Emissive	
•Rough	
•Transparency	
•Brilliance	
•Two Sided	
•Texture Selector	121
•Advanced Textures	122
•Normal Maps	124
•Environment Maps	124
•Texture Mode	127
•Material Layers	128

10. Scripts **... 129**

•Creating a Script	129
•The Script Directory	129
•Installing a Script	130
•Plug-in Scripts	130
•Parametric Plug-in Scripts	
•Export Plug-in Scripts	
•Installing Plug-Ins	131
•Running a Modeling Script	131
•Controller Scripts	131

•Script Errors	
•Writing Scripts	132
•Data Types	132
•Variables	132
•Expressions	132
•Function Calls	
•Unary Operators	
•Binary Operators	
•Statements	
◦Compound statements	
◦if statement	
◦while statement	
◦for statement	
◦User Functions	
•User Attributes	135
•Forum Notes: Sample Script	
◦ASL Functions to make glass	
11. ART Ray Tracer	... 138
•Art Materials	138
•Class anim8or	
◦phong	
◦reflection	
◦glossy	
•Class glossyreflector	
◦Kr	
•Class transparent	
◦IOR	
•Class dielectric	
◦IOR	

◦UnitDistance	
•Light Attributes	144
•sampler	
•Ambient Occlusion	144
•Ray Tracing and Anti-Aliasing	145
12. Rendering	... 147
•Rendering Images and AVIs	147
•Renderer	148
•Scanline	
•ART Ray Tracer	
•Alternate Renderers	
•Rendering Images	149
•Rendering your Image	
•Saving your Image	
•Anim8ors Image Formats	149
•BMP	
•JPG	
•PNG	
•Saving Your Images	150
•Channels	151
•The Alpha Channel	
•Image Compositing	
•The Depth Channel	
•Rendering .AVI Movies	153
•AVI format	153
•Rendering and Saving your AVI	154
•Scene Parameters	

Appendix A. Keyboard Shortcuts ... 158

- Menu Shortcuts 158
- Viewpoint Shortcuts 158
- Top Toolbar Shortcuts 159
- Object Editor Shortcuts 159
- Point Editor Shortcuts 160
- Figure Editor Shortcuts 160
- Sequence Editor Shortcuts 161
- Scene Editor Shortcuts 162

Appendix B. Standards and Aspect Ratios ... 163

- Image Standards 163
 - Graphic Image Standards
 - .BMP
 - .JPG
 - .PNG
 - Film Standards
 - 35 MM Film
 - Silent Film*
 - Talkies (pre-Academy)*
 - Academy*
 - CinemaScope*
 - Paramount VistaVision*
 - Techniscope*
 - Super 35*
 - 16 MM Film
 - Standard 16 mm*
 - Super 16 mm*
 - Ultra 16*

•8 MM Film	
◦ <i>Standard 8 mm</i>	
◦ <i>Straight 8</i>	
◦ <i>Super 8 mm</i>	
• Television Formats	167
•SDTV	
•EDTV	
•HDTV	
◦ <i>Formats</i>	
◦ <i>Frame Rates used for HDTV</i>	
• Analog Standards	168
•NTSC	
•PAL	
•SECAM	
•MUSE	
• Digital Standards	169
•ATSC	
•ISDB-T	
•DVB-T	
•DTMB	
• Aspect Ratios and Resolutions	170
•Paper	
◦ <i>Pixels and Dots</i>	
◦ <i>Common paper sizes</i>	
◦ <i>Common Print Sizes</i>	
•Common Resolutions	
•Film	
•Displays	

•Interlaced and Progressive

176

- 480i
- 576i
- 576p
- 720p
- 900p
- 1080i
- 1080p

Appendix C. an8format

... 177

- Overview
- Tokens
- Chunk Format
- File Layout
- Headers
- Description
- Environment
- Texture
- Material
- Object
- Figure
- Sequence
- Scene
- Misc. Items

Appendix D. ASL specifications

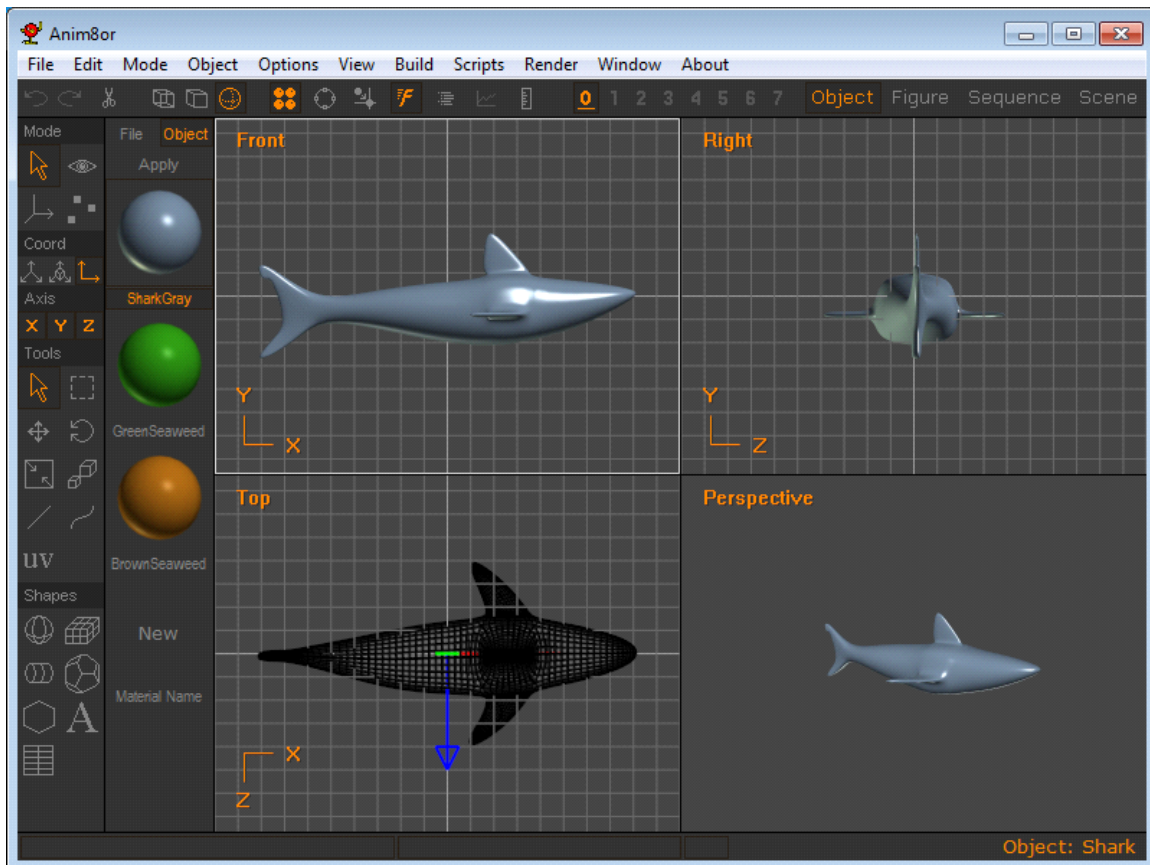
... 196

- Introduction
- Kinds of Scripts
- Variables and Names
- Data Types
- Comments
- Predefined Constants

•Variable Declarations	
•Expressions	
•Statements	
•Special Type Details	
•Predefined Variables	
•Functions	
•Script Kinds	
•Specifics of Script Kinds	
•Object Export Plug-in Script	
•Controller Expression Script	
•Debugging	
•Tracing Script Execution	
Appendix E. Learning ASL	... 226
•Forum Notes Special: (by member: polyGon_tError)	
Appendix F. Anim8or Scripts List	... 229
•Anim8or Scripts	229
•Parameteric Shape Plug-Ins	
•Object Export Plug-Ins	
•Sample Scripts	
•Scripts from Users	
•ASL Scripts Database	231
•Shape Scripts	
•Command Scripts	
•Export Scripts	
•Complete History of ASL forum Scripts	236
License	... 241
About The Author	... 243

Introduction

Anim8or is a 3D computer animation program that is designed to allow straightforward creation of animations. You interactively create and edit objects, figures, and scenes directly on the computer's screen. The basic interface is similar to most 3D animation and CAD programs, as shown in the following screen shot:



You control various aspects of your work using an ordinary computer mouse or tablet. You select and drag, rotate, scale and place objects by clicking in the various views of your work. There are two **toolbars** that you can use for common tasks. The one at the top of the screen has buttons for general commands that are used throughout Anim8or, while the one on the left allows you to change your mode of operation for common tasks in the current editor that you are using. The usual **menu** is there for less frequently used tasks.

You can control the view of your objects to see how they appear from the **front, left, top, back, perspective**, etc., and you can show either multiple views or a single view on your screen.

There are four editors.

First, there is an **object editor** that is used to build (mostly) static objects. This is where you

create and refine the basic models for your 3D world.

Second, there is a **figure** or **character editor**. You use it to define the structure of a character that you want to animate by connecting jointed **bones** together, and attaching objects to them. You can give each joint different ways in which they are allowed to move and restrict the range, just like your elbow or shoulder can move only in certain ways.

Third, there is a movement or **sequence editor**. Here you define segments of motion, such as one cycle of a walk. These can then be linked together in chains in the scene editor for longer sequences.

Finally, you put together your final scenes in a **scene editor**. Here you place the objects and built in the other parts of Anim8or into their final "world". You can control how they move, and where the **camera** is located. You can also give the scene various kinds of **lights**.

Once you have created a model or scene you can render high quality **.jpg**, **.bmp** and **.png images** and **.avi movies** and save them to disk for later use.

Finally, you can use Anim8or for 3D printed objects. You can import and export files in the **.stl** format used by 3d printers.

What's New in v1.0

This release represents a milestone in the life of Anim8or. The overall look has been updated to ease eyestrain that can result from long modeling sessions, and there are major new modeling and animation tools and workflow enhancements. Some of the highlights are:

- **A New look**
- **Inverse Kinematics**
- **Topographical Knife**
- **Fast Select**
- **Pressing the Alt key** temporarily changes into **Arc Rotate** mode
- **Layers**
- **The addition of Folders**
- **Additional improvements** and **additions** throughout Anim8or.

The major additions to Anim8or in version 1.0 for both the modeling and animation are summarized here and described in more detail in their respective sections throughout the manual.

User Interface

The user interface has several enhancements that can streamline modeling and animation. The **new look** eases eye strain, of which the most obvious is the new color scheme which has a **dark gray background** instead of light blue. This makes models stand out from the background so their shapes are generally easier to see. You can also increase the size of the toolbar buttons which helps touch-screen users. Additionally, with an adequate graphics chip, you can change

the 3D views to **anti-aliased**, and show edges and controls in **thicker lines**, which can make them easier to see and so the edges don't disappear on higher resolution screens.

Enabling **fast-select** allows a more fluid selection and editing work flow since it eliminates the need to continually switch modes to edit different parts of a mesh or scene without changing to the Select tool. Moving the mouse highlights in orange the component that would be affected if you click making it easier to select the correct point or edge. This can be disabled for beginners who sometimes find it easier to separate these functions

Layers are an addition that lets you assign components to different groups that can be locked or hidden by the click of a button.

You can temporarily enter **arc-rotate** mode by pressing and holding the **Altkey**. **Arc Rotate** mode allows you to model fluidly within an model or scene while editing. Similarly the mouse wheel zooms your view around the position of the cursor.

Modeling

There are several improvements to the modeler. The most important is the **topographic tool** or **topo-knife**. It is a 3d modeling tool that supports flexible, rapid editing of meshes. You can move points or edges, slit and join faces, add and "dissolve" edges and points, and much more with this versatile " all in one " tool.

*Give a big thanks to **Raxx** for helping me immensely in the implementation.*

Scaling and rotating faces, edges and points now does so relative to the center of the group that's selected, rather than the center axis of the mesh.

You can assign objects to different **layers** that you can lock or hide as a group.

The addition of **folders** for objects, figures, bones, etc. helps organize large projects into more manageable pieces.

There are additional improvements and additions throughout Anim8or. These include **normal maps**, improved **face extrusion** for corner and right angle edges, **scale** and **rotation** that apply to the center of selected parts, and more.

Animation

The main addition to Anim8or's animation capabilities is **Inverse Kinematics**, or **IK**. This allows rapid animation of rigged characters. With it you position a chain of multiple bones at the same time. You simply position the end of the one bone where you want it and all of the other bones in the chain adjust and they all move as needed. You can also lock a foot to the ground, move the body and the leg will adjust as needed.

Features

- 3D Modeler - Create and modify 3D models. Built-in primitives such as spheres, cylinders, platonic solids, etc.; mesh-edit and subdivision; splines, extrusion, lathing, modifiers, bevel and warps.
- TrueType font support - 2D and 3D extruded text for any TrueType font.
- OpenGL based real time operation.

- Import and modify .3DS (3D Studio), .OBJ (Wavefront), and .STL (3D printing) object files,
- Export .3DS, .OBJ, and .STL files,
- Jointed character editor,
- Inverse Kinematics,
- Morph targets,
- Anti-aliased software renderer for high quality, production quality images,
- Create 3D scenes and animations and output .AVI movie files, .JPG, .PNG and .BMP images,
- Supports textures, bump maps, soft shadows, spotlights, fog, and much more,
- Texture support for .JPG, .PNG, .GIF, and .BMP format files,
- OpenGL shaders for realistic previews.
- Scripting language.
- Plug-ins for parametric shape, object export, editing tools.

System Requirements

- Windows WinXP, Vista, Windows 7, 8 or 10.
- OpenGL accelerated graphics card or integrated graphics,
- 512 MB memory, 1 GB recommended.
- 25MB disk space.

Linux: Using Anim8or on Linux

All Linux fans will be glad to know that that you can run Anim8or too. It reportedly works quite well on Linux using WINE.

Installation Instructions

Anim8or is very simple to install and run. It is composed of a single executable file that can be put anywhere on your disk. I like to keep things simple.

- Download the latest version of the Anim8or program in the file **animv100.zip** from the website.
- Unzip it into your target directory. (**Zip Files:** The Anim8or executable file and manual are stored in a compressed form called a **.zip** file. In this form they are much smaller than their normal storage format so it takes less time for you download them (and I pay less for bandwidth). You can't use the files until you restore them to their full size. Many of the newer operating systems now are able to unpack a .zip file, however you can do this with a nifty program called **WinZip**. If you need a copy of WinZip you can download a free demo version from the website www.winzip.com.)

- Run the file **anim8or.exe**. (No installation required.)

That's all there is to it.

Note: The previous releases, are still available if you run into any problems with the new release.

News:

An update to Anim8or, **v1.00b**, is available with a few bug fixes. The link can be found in the header of the Forums pages. Included in the **v1.00b** zipfile download you will find the file **Anim8orSplash.dll**. It allows you to see different splash screens when you run Anim8or. Copy the file Anim8orSplash.dll to the same directory as Anim8or.exe. To use your own splash screen, place an image file named splash.png or splash.jpg in the "images" director (see the File->Configuration dialog).

The current development release is **v1.01.1318**, January 25, 2018 which can be found in the forums if you want to try it out.

Compatibility

V1.0 should be able to read previous versions of .an8 files. Though the opposite is not entirely possible, it generally should work especially for the most recent releases. Newly added features will be ignored, of course, but shouldn't cause other problems.

File Formats

Anim8or can load, save, import, and export several types of file formats including both ASCII and binary formats of STL used in 3D printing. By using plug-ins Anim8or has the ability to use even more.

- *Load and Import:*

Anim8or (**.an8**), 3D Studio (**.3ds**), STL (**.stl**), Lightwave (**.lwo**), Wavefront (**.obj**), DirectX/D3D (**.x**) (plug-in : no animations)

- *Save and Export:*

Anim8or (**.an8**), 3D Studio (**.3ds**), Wavefront (**.obj**), Vertex (**.vtx**), C source file (**.c**), STL (**ASCII**) (**.stl**), STL (**binary**) (**.stl**), DirectX/D3D (**.x**) (plug-in : no animations), VRML (**.wrl**) (plug-in), Roblox (**.mesh**) (plug-in), Orbiter Mesh (**.msh**) (plug-in), Celestia (**.cmod**) (plug-in), Game Maker 6 (**.gm6**) (plug-in), WRL (**.x3d**) (plug-in)

Scripting Language Specification

As of v0.95 Anim8or supports a scripting language currently called ASL for "Anim8or Scripting Language". ASL allows more flexible controllers, scripted object modeling, and notably plugins. You can add new formats for file export and define new parametric shapes with ASL scripts.

The basics of using scripts is described in this Manual. If you want to write scripts, the specification for ASL is here also. There are many scripts on the website that are available and you are able to open them in Notepad or another ASL editor in order to study them to assist you in writing your own.

There is also a forum for script discussions. If you find any problems with the specs or in Anim8or when running scripts please let me know. If you have questions about how to write or use scripts, please post them in the Anim8or Scripts forum.

*Installing Scripts and Plug-Ins is described in **Chapter 10. Scripts.***

Anim8or Tutorials

On the website you can find tutorials that are designed to show you how to use Anim8or, whether you are a beginner or an experienced 3D animator. You will find that many techniques are similar to those used in other programs. But 3d animation is an inherently complex endeavor. And minor differences between different programs can be a source of frustration. I hope that these tutorials may be of some use to you.

I strongly suggest that you work through the *Eggplant* tutorial, especially if you are new to 3D modeling. It is designed to introduce you to the concepts of 3D modeling, and especially to how Anim8or modeling works. For an introduction to animation, you should do the *A Simple Walk* tutorial.

Anim8or Support

You can find support for Anim8or from several places:

- See the Anim8or v1.0 Discussion Forum
webpage_ <http://www.anim8or.com/smf/index.php/board,8.0.html>
- You can also ask a question in one of the other Anim8or **forums**. By doing a search of the forums you will find that many common questions have already been answered there.
- E-mail me at **support (AT) anim8or.com** I'll answer as as soon as I can but it may not be right away. It will help if you put ANIM8OR in the subject line. I get a lot of spam every day. Mentioning Anim8or in the title will help make sure that I don't accidentally miss your message.



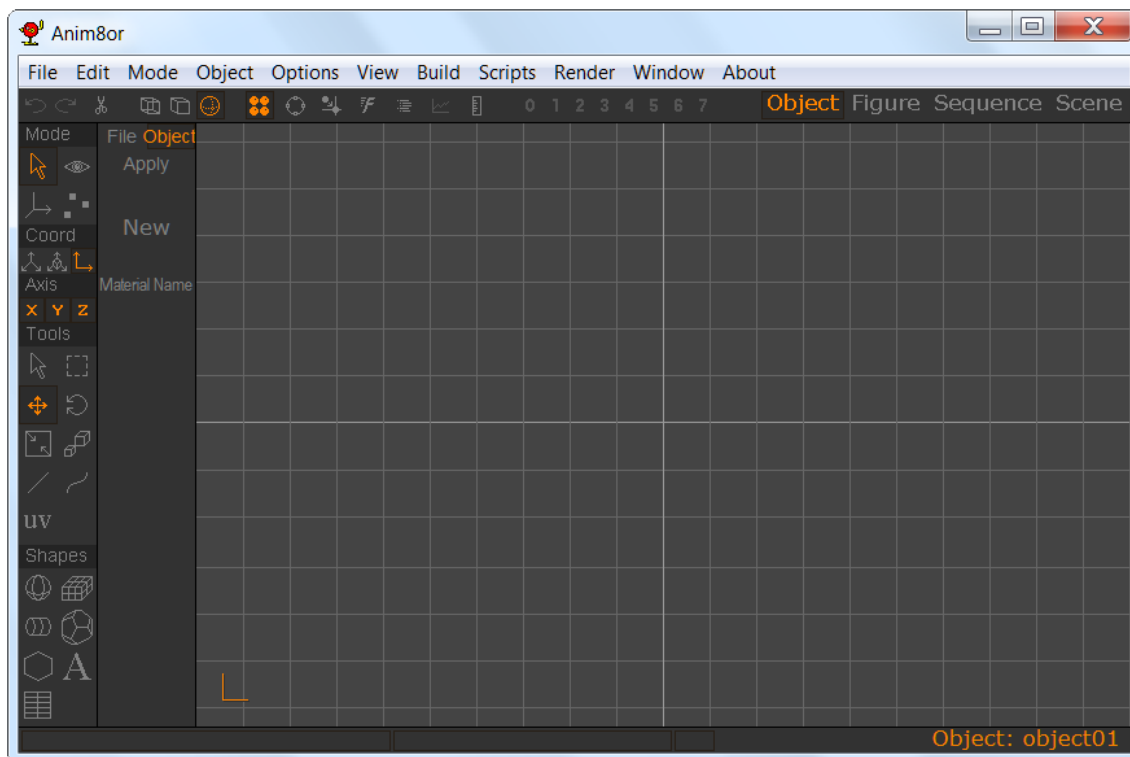
Anim8or® is a registered trademark of R. Steven Glanville.

Copyright 2018 R. Steven Glanville

Anim8ors' New User Interface

The first thing you notice is the new look of the **User Interface**. It's dark shark gray theme complements the new features and makes it easier to work within Anim8or.

However for those of you wistful for the old nostalgic look it is still available. You can change between the new look and its older predecessor by going to **File→Configure-UI** and unchecking the default setting **New Color Scheme**.






User Interface

Menu and Toolbars

Menu

The Menu Bar at the top is the mainstay of your project from beginning to end. The drop-down menu buttons and their sub-menus are where you find all of your base commands to be your roadmap and tools while bringing your ideas and dreams to life.

You will find that various aspects in and on the menus will change depending on the mode that you are working in or whether an option is open to you. If an option is available the text will be in black. Grayed-out options are not available. In those that are available, clicking on them will

perform their action, check or uncheck a command  , open a new window for your input  , or take you into a sub-menu  . Most of the menu button options are explained throughout the manual in their appropriate sections.

Modes and their menus

Object Mode: The first screen that Anim8or opens onto after the splash page. Here is where your objects are built for your final scene.

Object Mode Menu

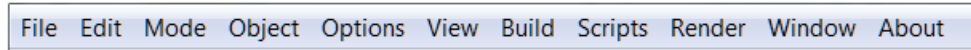
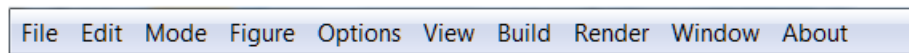


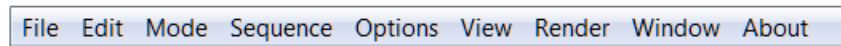
Figure Mode: Here is where you build and connect a skeletal structure to your object (the connecting process is known as skinning) so that it will be able to be animated beyond simple movements.

Figure Mode Menu



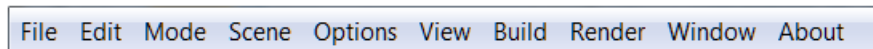
Sequence Mode: Sequencing is merely designing and implementing little motions that will later be combined together in your final scene to bring your final creation fully to life.

Sequence Mode Menu



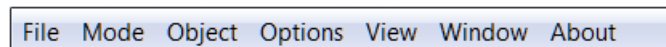
Scene Mode: Together with your objects and moveable figures you will be able to incorporate lights, shadows and effects to be rendered into your finished work as images or movies.

Scene Mode Menu



Browser Mode: Opens up to show you a visual library bar along the right side. Here you will find each of the objects in your project and you can select them to check them over in your viewports. You cannot edit or affect your elements in any way. The mouse buttons work similar to Arc Rotate so that you can view them from various angles and distances.

Browser Mode

















Top Toolbar

This is the toolbar that's located at the top of the window beneath the Menu. You can use it for several tasks common throughout Anim8or.

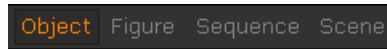


The buttons on the top toolbar do the following things:

-  **Undo [Ctrl-Z]** - If this button is enabled you can use it to undo the most recent edit command.
-  **Redo [Ctrl-Y]** - Reapply a command that you have un-done.
-  **Cut [Ctrl-X]** - Deletes any selected components and places them on the clipboard.
-  **Wireframe [Ctrl-W]** - Changes the currently active view of your object to Wireframe.
-  **Flat [Ctrl-F]** - Changes the view to Flat or Faceted.
-  **Smooth [Ctrl-S]** - View objects as smooth.
-  **Material [Ctrl-M]** - Toggles the display of the Material toolbar.
-  **Arc Rotate [Ctrl-R]** - Enables the Arc Rotate tool. You can easily pan, zoom, and rotate any view with Arc Rotate. It is described in the next section.
-  **Snap to Grid [Ctrl-G]** - You can toggle Snap to Grid mode with this button. The grid tool has two behaviors. Normally it snaps points to unit coordinates (i.e. multiples of the current grid setting - assume it's set to a grid snap of 1.0) relative to the **shape's** coordinate system. If they are originally in a circle some of the coordinates will have initially fractional values. After snapping they will all have integer values in the shape's coordinate system , so won't be in a perfect circle. However if you hold the Shift key down then the points will move in integer steps **relative to their original position** and thus will keep the circular shape.
-  **Fast Select [Ctrl-T]** - When this is enabled you can click on unselected objects and immediately edit them. Experienced users often prefer this mode because it allows faster editing, while newer users find it easier to first select, and then edit things.
-  **List Items [Ctrl-L]** - This button will open a text window listing the objects in the current view. You can click on them individually by name to select or deselect them.
-  **Graph Editor [Ctrl-P]** - This button will open a window with a graphical view on the keys in your Scene or Sequence. You can use the mouse to edit keys here.
-  **CAD Notations [Ctrl-D]** - Displays additional measurement notations on the selected object.
-  **Layers** - There are 8 editing layers which can be locked or hidden. You can assign individual components to any one of the 8 layers. Clicking on the layer button toggles that layer's visibility. If the button for the layer is grayed out then it is hidden, and dark gray layers don't contain anything. Right-clicking locks and unlocks a layer, indicated by a small lock next

to the number. Objects in locked layers are visible but you cannot select or edit them.




The right portion of the top toolbar has four buttons that allow you to quickly switch between different editors:



You can also use **Shift+RightArrow** and **Shift+LeftArrow** to move between editors.

Side Toolbar



Here is a portion of a typical toolbar. It has several sections in it that control different aspects of its behavior. The first thing that you will notice is that markings on some of the buttons are in **ORANGE** , some are in **GRAY** , and a few are in **DARK GRAY** . The orange ones indicate that the associated mode or option is currently selected or in effect. The gray ones indicate a state that is not selected. The dark gray buttons show options that are disabled.

The top group of buttons on each toolbar act as a radio set indicating the current editing mode of the active view. When you click on any one of them, it becomes highlighted and that mode is selected. All of the others are deselected. The meanings of the various buttons in this group, and the other common buttons, are described below.

Common Button Meanings

The topmost group can have up to 4 buttons in it. When you click on one it sets the indicated mode, and changes the lower part of the toolbar to show the functions that are present in the new mode. The different modes are:



[A] This button indicates that you are in the most basic select-and-edit mode for the editor that you are currently using. You will use it for most operations, such as selecting things and common editing actions.




[V] This button is used for changing your view or views of your workspace. With it you can pan, zoom, tilt, and scale each view independently.




[O] In this mode you can move and rotate the pivot for objects in your workspace. A pivot is the point and orientation that are used when you scale and rotate each object.



[P] You use this button to move into point edit mode. It allows you to add, move, and modify individual points, edges, and faces of editable mesh objects.

The next row of three small buttons  shows the three axes of the 3D world. They indicate which axes are "unlocked", allowing you to move and rotate things in each direction. You can individually select or deselect them, and thus restrict the movement of objects that you are changing.

Below these buttons you will find three more small buttons  of which only one can be active at a time. They show the current coordinate system you are using. You can use **world**, **object**, or **screen** coordinates when you manipulate things.

You will find that there are several common toolbar buttons that have the same or similar meanings in more than one editing mode. They are described next.



Select [*a*] - You use the arrow button to select individual components. In this mode clicking on an object with the left mouse button selects it and deselects previously selected objects. Right-clicking selects additional objects without deselecting currently selected ones. **Tip:** holding down the **Ctrl** key then pressing the mouse button allows you to select multiple objects using the mouse as a sort of paintbrush.



Drag-Select [*d*] - With this button you can click and drag to show the corners of a selection rectangle. Any and all objects that fall within that area are then selected. Again, the left mouse deselects any previously selection items, while the right mouse does not.



Move [*m*] - This is the move function. You set it when you want to move your objects around on the screen. When you press the left mouse button within a view window you can drag anything that is currently selected in the X and Y directions by moving the mouse to the left and right, or up and down. The objects will move along with your mouse. Similarly, the right mouse button will move them along the Z-axis, which is usually into the screen.



Rotate [*r*] - This is the rotate button, used for rotating things. When you click and drag the left mouse button, any selected objects will rotate around the X and Y-axes along with your mouse. You can use the right button to rotate them around the Z-axis.



Non-uniform Scale [*n*] - This is the non-uniform scaling button. With it you can scale most objects independently in the X any Y directions using the left mouse button, and in the Z direction with the right mouse button.



Scale [*s*] - This button scales objects in all dimensions simultaneously.



Straight Path [*l*] - This button allows you to create a straight line between two points.



Curved Path [*p*] - This lets you create a curved line useful for creating paths and shapes that can be used for lathing an object.

In the Sequence and Scene editors there is set of buttons for controlling the playback of your animations. You use its buttons in the same manner as those on a video player.



Viewports

The main viewing area is where you find the viewing ports. These can be accessed from **Menu→View**. Depending on which mode you are working in, you may find some of them are not available.

You can use anywhere between one and four viewports and those can be chosen from the drop-down menu list. They are as follows: **All** - four equally sized viewports. **2 Views** - two equally sized viewports (this is the default for stereo view). **3 Views** - one larger main viewport and two smaller ones to the right side. **1+3 Views** - one larger viewport and three smaller ones located at the right side. By clicking on and highlighting a viewport you will begin working in that view and still see your results in the other ones.

While the names of the views are mostly self-explanatory below is a brief description of each:

View	Build	Render	Window	Alt
<input checked="" type="checkbox"/>	Wire		Ctrl+W	
	Flat		Ctrl+F	
	Smooth		Ctrl+U	
	Preferences ...			
<input checked="" type="checkbox"/>	All		4	
	2 Views		2	
	3 Views		3	
	1+3 Views		5	
	Front		Num-5	
	Back		Num-7	
	Left		Num-4	
	Right		Num-6	
	Top		Num-8	
	Bottom		Num-2	
	Ortho		Num-9	
	Perspective		Num-3	
	Stereo			
	Camera		Num-1	
	Other			▶
	User			▶
	Frame		f	
	Frame Selected		F	
	Z Limits			
	Reset			
	Reset All			
	Default Visuals			

Front	- view from the front (looking towards +z on the z-axis)
Back	- view from the back (looking towards -z on the z-axis)
Left	- view from the left side (looking towards +y on the y-axis)
Right	- view from the right side (looking towards -y on the y-axis)
Top	- view from the top (looking towards -x on the x-axis)
Bottom	- view from the bottom (looking towards +x on the x-axis.)
Ortho	- orthographic view (contents of the viewport are drawn using parallel lines so there is no visual distortion.)
Perspective	- perspective view (contents of the viewport are drawn using converging lines from the viewer to a point or points in the far distance. This is representative of how we view things with our vision in that farther things appear smaller and there is a foreshortening of objects in the environment.)
Stereo	- two side-by-side viewports, each representing the view as seen by one of our two eyes from their respective locations. Our brain interprets this as a 3D image when viewing the two images in such a way that they overlap each other.

Camera	- view as seen from the camera. The green square represents the main area that will be rendered for images or animations.
Other	- usually the view from an additional object (such as a second or third camera)
User	- a user defined view for a specific purpose wanted or required for any given project.

Mouse Usage

Anim8or requires the use of a mouse with two buttons, but using one with a middle button can simplify some commands. You use the left one most of the time to select, move, and change objects. When selecting, the **left button** always deselects everything beforehand, so that only a **single item** remains selected. The **right button**, on the other hand, allows you to select **multiple items** by adding newly selected ones to those already selected. You can use the **middle button** to **deselect** specific objects. If you don't have a middle button some operations will require a few more steps.

When moving, rotating, or scaling something non-uniformly, the **left button** changes the **X-axis** and **Y-axis**. Moving to the left or right affect the X-axis, and up and down the Y. The **right button** changes the **Z-axis**.



You can usually **double click** on something to bring up a dialog to view and edit its properties.

Keyboard Shortcuts

Many tools and commands are associated with **keyboard shortcuts** or **hot keys**. These are shown in this manual enclosed by square brackets [**L**] next to the button or command. You can press this key instead of clicking on a button or selecting a command from the menu. They are summarized in **Appendix A: Keyboard Shortcuts**.

Undo and Redo

Anim8or has a complete **undo** command for the Object, Figure, Sequence and Scene Editors. As long as you stay in the specific Editor you can **undo** and **redo** multiple editing commands. A history is kept of as many changes as can fit in the undo buffer, initially set to about 1 MB of memory. You can increase this in the Configuration dialog (25 MB was default previously). So unless you're editing large models, you will be able to undo more than one operation.

You use the **Edit→Undo** and **Edit→Redo** menu commands to apply undo's and redo's. Or you can use the top toolbar buttons  [**Ctrl-Z**] and  [**Ctrl-Y**].

If you wish to cancel a currently active operation, such as if you are dragging the mouse to change an Object's orientation, you can still use the original (rather simplistic) substitute. When you are dragging a mouse in the main window to change something, say an object's orientation, and you think "Oops! I didn't want to do that!" just keep on dragging the mouse until it's outside the active view window and release the button. This will cancel the operation.

Still, when using this and any other computer program, I still offer the same advice: "Save Early, Save Often!" And don't always save to the same file. Keep a recent backup handy.

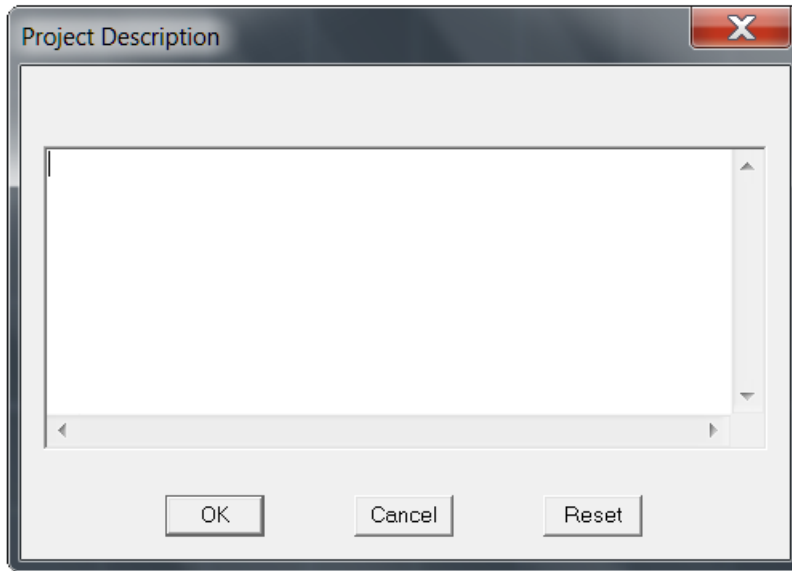
Tool Tips

You can turn tool tips on with the **Options→ToolTips** menu command. Then if you pause the mouse cursor over a toolbar button its function will be shown in the status bar.


Project Description

In order to help you keep your notes handy and within your project Anim8or has a built-in note utility. Here, you can keep your project description, copyright statements, any ideas, problems you've stumbled upon, or any noteworthy information that you wish to save. It is included in

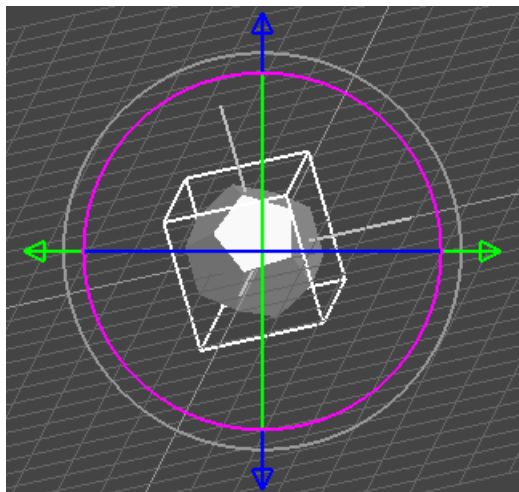
your saved project file and will allow you to refresh your memory as to pertinent information. This is accessed via **Window**→ **Project Info** in the menu.



Arc Rotate


The directions for using the new Arc Rotate has changed in **Anim8or 1.01.1318**. By holding the Alt button you can use Arc Rotate temporarily. By clicking on the  button in the top toolbar you will be in Arc Rotate mode until you turn it off.

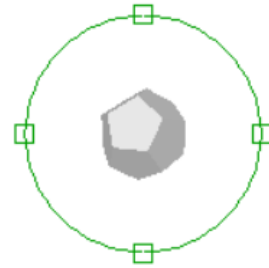
Click-dragging **any** button on the green, blue, or magenta lines (representing the x (green), y (blue), and z (magenta) coordinates) in the inner circle rotates the screen about your objects pivot point to view it from different positions. If you Click-drag using **any** button on the blue or green arrows coming from the outside gray circle it will move everything in the viewport you are using in the **up-down** or **left-right** directions. To get back to your original view just reset the viewport by going to the appropriate button under **View** in the top Menu bar or use the listed Hotkey.



Arc Rotate in Anim8or 1.01.1318

The directions below have been retained here for backwards compatibility with the older versions. The original Arc Rotate is still available in Anim8or 1.01.1318 and can be found by going to **Options**→ **Debug**→ **Use Original Arc Rotate**.

When you press and hold the **Alt** button Anim8or temporarily enters **arc rotate** mode and displays a special overlay over the focus window. You can then use the mouse to pan, scale and rotate your view. It may take you a try or two to get used to how it works, but once you have it can save you a lot of time over other methods. You can also toggle arc rotate by clicking on the arc rotate button  [**Ctrl-R**] in the top toolbar.



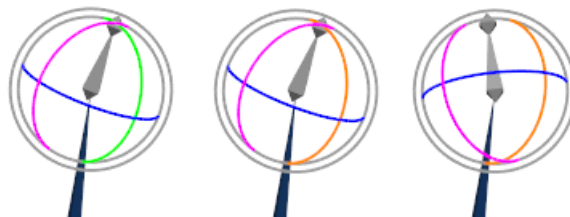
Also previously:

In the view of the arc rotate screen there are six different regions on the screen, separated by the green overlay: the inner, outer, top and bottom, and left and right. Each behaves differently when you click your mouse in them, and each mouse button controls a different function.

- Left button - Rotate the view. Clicking in the center rotates it around the X and Y-axes. The outer region rotates around the Z-axis. Clicking inside one of the four little squares rotates in the X-axis (for the top and bottom) or the Y-axis (for the left and right) only.
- Right button - Dolly the view. The center area moves your view left and right, and the outer region moves into and out of the screen. You may not see any change when dollying into the screen in orthogonal views but they will be apparent in a perspective view.
- Middle button - Scale the view. The different areas all behave the same when you are scaling a view.

Editing Widgets

New to version 1.0 of Anim8or, **widgets** are on-screen controls that both simplify and offer better control of changes you make. Shown below is a trackball that is used to rotate bones around individual axes. The colored arcs show you which rotations are available. Green represents a rotation around the X-axis, blue the Y-axis, and magenta the Z-axis. Moving the mouse over the highlights the available controls.




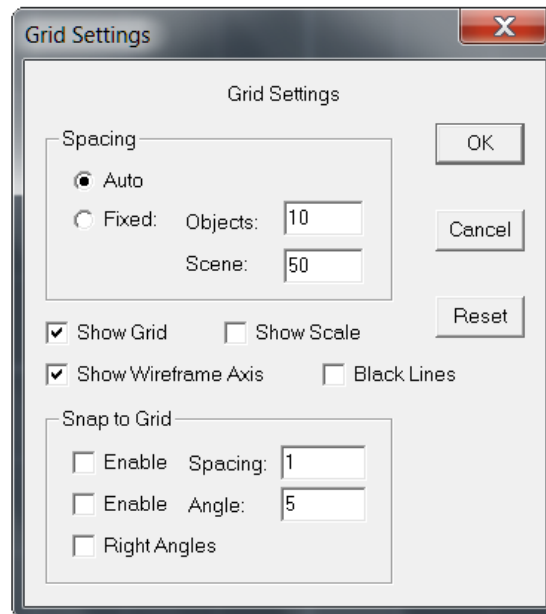
Grid Control

You can set the size of the background grid used in the editing views, or you can let Anim8or pick a default size that changes with how much you zoom the view so that it always shows a reasonable number of grid lines. Use the **OPTIONS→GRID** command to show the grid dialog.

There are two grid sizes that you can set. One is used in the Object, Figure, and Sequence Editors, and the other is used in the Scene Editor.

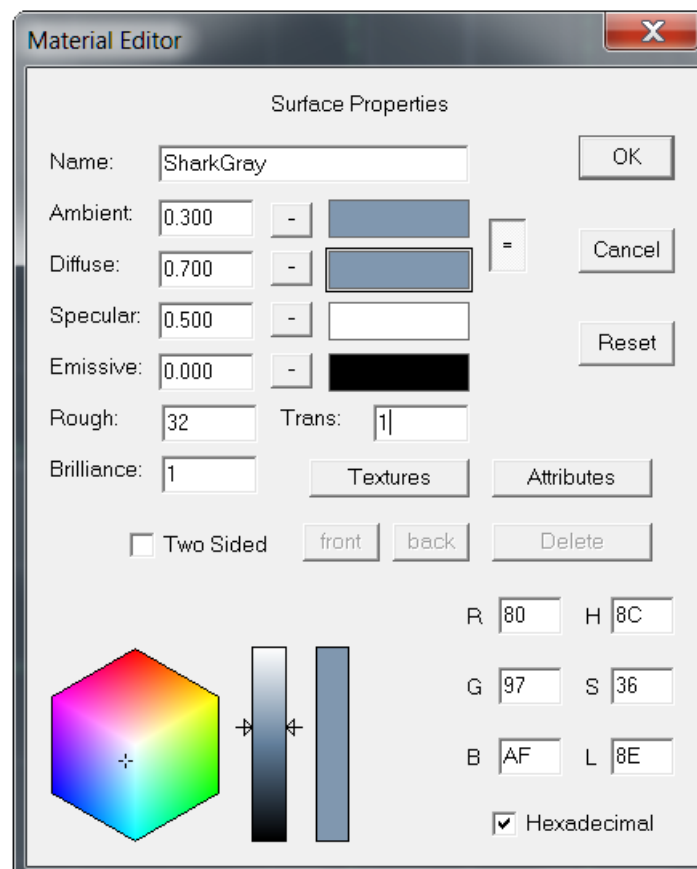
You can also set snap-to-grid for the Object editor by checking the Enable box in the Snap to Grid section,

or with the  snap to grid button [**Ctrl-G**] in the top toolbar. This will position objects you move at locations on a grid, making it easy to align them. Snap-to-grid also controls the size and rotation for many operations to give you more control on other aspects of modeling. You set the size of the grid snap in the Spacing entry. The Right Angles box is useful for adding edges at precise 90 degree angles.



Material Editor

The material editor is used whenever you need to design the color, texture, transparency, and other visual properties of an objects appearance. An example is shown below:



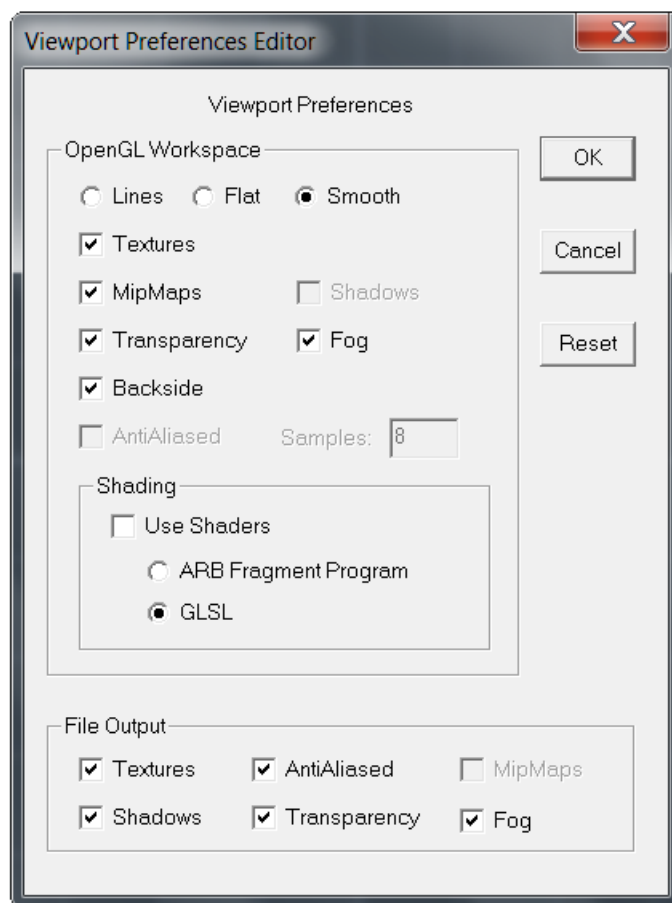
Materials are a complex topic. This dialog box has several sub-dialogs for certain functions. See **Chapter 9: Materials** for more details on how to use the Material dialogs.

Anim8or Object Libraries

You can use any Anim8or **.an8** project file as a library. Objects, Figures, Sequences, and Scenes can all be imported to another Anim8or project with the **Object→Import**, **Figure→Import**, etc. menu commands. You can also export individual Objects and such with similar **Object→Export** commands. Make sure to select the **.an8** file type.

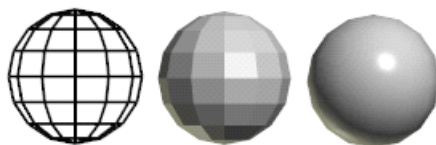
Visual Quality

You can control the quality of the images shown when you edit, as well as the quality of the images output to files. These are controlled through the Viewport Preferences dialog that is found under the **View→Preferences** menu command. It looks like:



OpenGL Workspace

This section controls the appearance of your interactive workspace. You can view your objects as wireframe models, as flat faceted solid objects, or as smooth shaded solid objects.



Most graphics cards display smooth shaded images at the same speed as flat shaded ones, so you might think that flat shading wouldn't be very useful. But it is sometimes helpful to see individual facets on a model.

Texturing, and mipmaps, can sometimes slow the response of your computer, so use the check boxes to enable or disable them as you see fit. Also, texturing can hide the structure of your models, so you don't always want to see it.

You can add a general **fog** to Anim8or Scenes. Objects beyond a certain distance gradually fade into the color of the fog. Fog is only visible in the camera view of the Scene editor.

Most computers support **anti-aliased** 3D graphics directly in hardware. Checking this box changes the display to use that mode. You may need to restart Anim8or for to be effective.

The **Use Shaders** check box enables drawing higher quality materials in the working views. It is available only on computers that support OpenGL pixel shaders. Graphics cards that support DX9 or higher will usually work. If your graphics card doesn't have this feature the check box will be disabled.

File Output

This section controls the appearance of your interactive workspace. High quality images are rendered using a software renderer or a ray tracer instead of an OpenGL accelerator, so the speed of your file renderings can be affected quite severely by the settings that you choose.

Anti-aliasing helps to remove jaggies but takes several times longer to do.

User Attributes

You can add your own unique numeric and non-numeric properties called **attributes** to your models, materials, characters, etc. Attributes are animatable. While they are primarily useful in scripts, they are quite flexible and are sometimes used to help develop new features for Anim8or. Attributes are saved in the .an8 output files so they can also be used to share data with auxiliary programs that read the .an8 format. See **Chapter 10 - Scripts** for more details on attributes.

Saving Images as .jpg, .png and .bmp Files

You can make a quick preview image of any view that you are working on with the **Render→Render Image** command (in the older versions: File→Render-Preview). You can also save this image to a **.jpg**, **.png**, or **.bmp** file, or print it, from within this command.

Printing

Anim8or has the usual Print and Print Preview options in the File menu. But it offers one additional level of control of your printed images, by letting you control the size and other properties of the bitmap image used in printing. It is not at all uncommon for a color printer to have a resolution of 1440 by 720 dpi. But generating an 8.5" by 11" image at this resolution is not practical! Instead Anim8or uses a **default** size of **1024x768** which you can adjust to suit your needs.

Auto Save

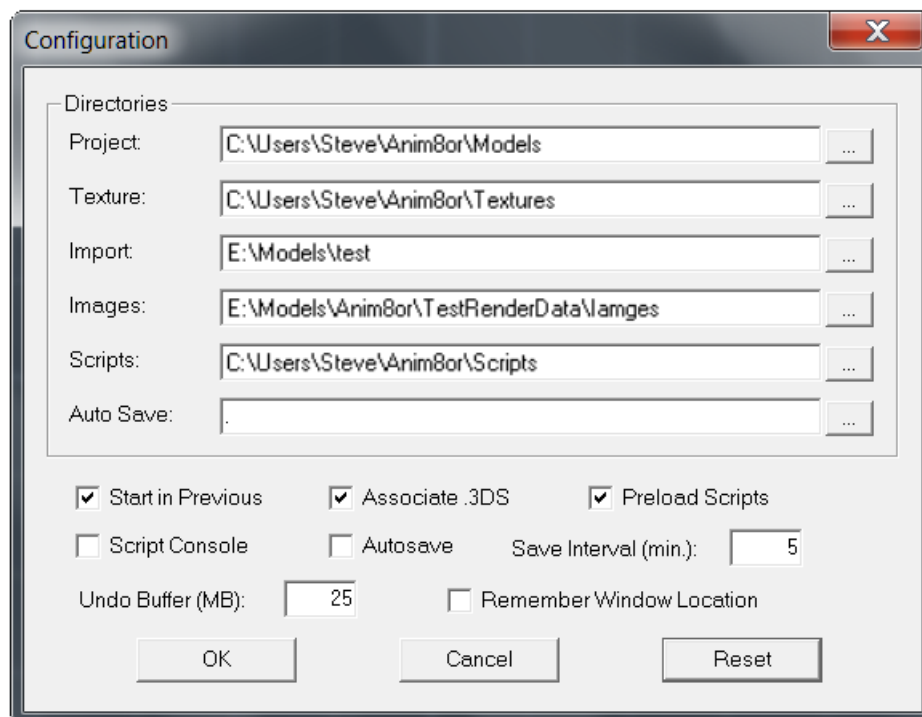
There is an Auto Save feature in Anim8or that will save a backup copy of your project at regular intervals when the project has been modified. You can set the interval, which defaults to 5 minutes, in the **File→Configure** dialog.

The name of the file is `_autosave000000000.an8` or something similar. You can set the directory that the back-up file is written to as well. The default is the current working directory.

Anim8or does not automatically try to restore the file after a crash, or notify you that one exists. But remember to look for one when you've had a problem.

Configuration

Anim8or keeps track of several base working directories. One is used to store Anim8or project files (.an8) and the others are the default directories for keeping textures, importing and exporting objects, saving pictures, and to store scripts which you can use to customize Anim8or in various ways. You set these directories with the Configuration dialog through the **File→Configure** command:



When you load or save a project file Anim8or uses the **project directory** as the initial default. You can change this directory in the Configuration dialog.

When you load a project file, Anim8or looks for texture files in three directories. First it looks in the same directory as the .an8 file. If that fails it tries the **texture directory** set in the Configuration dialog. As a last resort Anim8or uses the full path name stored in the project file, which is where the texture file was initially found.

The **import directory** is simply the default place for importing and exporting files of any format.

The **images directory** is the default location for saving any images or movies that you render.

The **scripts directory** is where Anim8or looks for scripts to preload. This is where you keep plug-ins as well since they are scripts. Scripts are only preloaded if the Preload Scripts box is checked.

Anim8or V 1.0 build 1.01.1318 Manual

checked.

You can have Anim8or automatically load the last project you were using by checking the "Start in Previous" box.

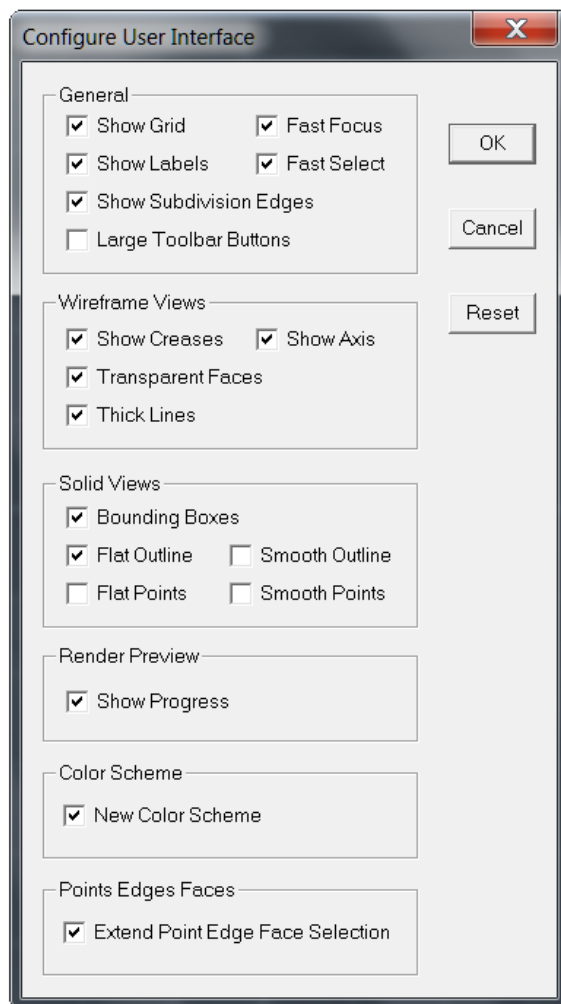
You can enable **Auto Save** by checking the box labeled accordingly and set the time interval of the saves.

If you check the Associate .3DS box then you will be able to start Anim8or by double clicking on .3ds files in Windows.

User Interface Configuration

You can change certain aspects of the way that the Anim8or working views look with the User Interface Configuration dialog. You can find it under the **File→Configure-UI** command:

Most entries have obvious meanings but you might need an explanation for a few of them.



Fast Focus only applies when you have multiple working views visible at the same time. It controls what happens when you click inside a window other than the one with the mouse focus. If fast focus is enabled then the current active editing command will be applied to the new window on the first click. So if you are in Select mode then any object you click on in the new window will be immediately selected. If fast focus is disabled then the first click will only change the focus window. You will have to click a second time to select something.

Fast Select allows you to both select and edit components with a single mouse click. It can be somewhat difficult to use for newcomers to Anim8or. If you disable Fast Select you will generally need to change to one of the select tools to select components, and then back to the tool you want to use, but you are unlikely to accidentally deselect your work by mistake. You can also hold down the **Shift+Ctrl** keys to temporarily switch to Arrow-Select mode.

Large Tool Buttons increases the size of the buttons in the toolbars. Users that have a touch screen often find this helpful.

Transparent Faces in the Wireframe Views section controls the drawing of a thin, transparent skin on the faces of a model in wireframe viewing mode. This is on by default but you may sometimes find it easier to edit a particular mesh with it disabled.

You can show the wireframe **outline** or the vertex **points** overlaid on your solid models in both the flat and smooth viewing modes by checking the appropriate boxes in the Solid Views area. You may find that showing all this information can slow your computer down, especially if you

are editing complex models, or make your model difficult to see. If this happens try disabling these features to improve performance.

Thick Lines. On very high resolution screens the lines used to show mesh edges and control structures can be difficult to see. Enabling Thick Lines changes makes them wider and easier to see. Enabling anti-aliased viewports as well can dramatically improve the appearance of the 3D workspace.


You can show the wireframe **outline** or the vertex **points** overlaid on your solid models in both the flat and smooth viewing modes by checking the appropriate boxes in the Solid Views area. You may find that showing all this in-formation can slow your computer down, especially if you are editing com-plex models, or make your model difficult to see. If this happens try disabling these features to improve performance.

When you render an image Anim8or normally updates the screen to show the progress of the render. Some graphics cards are very slow when updating the screen and this can increase the time it takes to render an image considerably. You can disable these partial image updates by unchecking the **Show Progress** box in the Render Preview section. Anim8or will still display a progress bar so you will still see how far the render has progressed. The final image will still be displayed when it is finished.

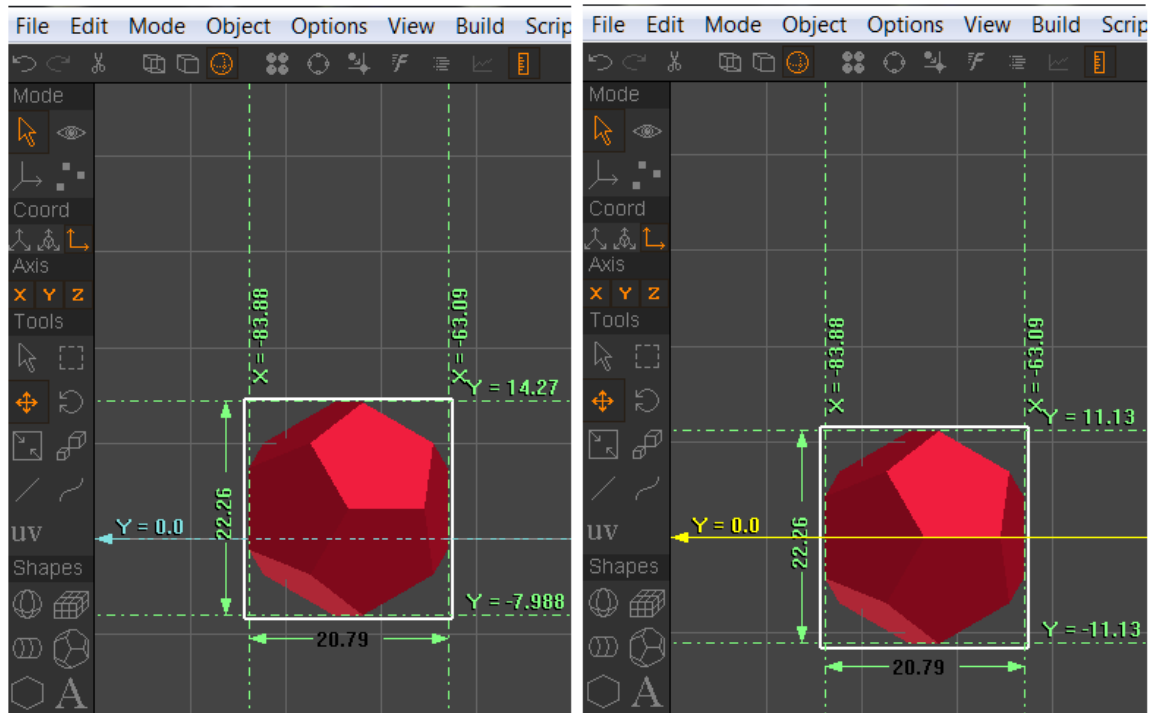
CAD Notations

When **CAD Notations** [*Ctrl-D*] is turned on it displays additional measurement notations on the selected object.

Alignment Ruler: The CAD ruler now has some preliminary alignment properties. You can align shapes along their top, center and bottom with the Alignment Ruler. Note that the alignment ruler snapping supercedes the grid snapping. It also snaps to grid when Grid Snap is turned on.

- Make sure CAD mode is enabled by pressing the  button.
- Click on the Alignment Ruler to enable/disable alignment. This is the little arrow at the left or bottom of a view.
- Click drag the arrow at the end of the ruler to move it. Note that the alignment ruler uses the Y-axis in the viewing windows.
- If you have one shape selected, the ruler will snap to it's top/center/bottom when you are within the current Grid-Snap distance. The color will change from **Blue** to **Yellow** to indicate that it has snapped.
- If you move a shape when the ruler is selected it will snap to the ruler in a similar manner.
- The ruler only appears in the **front, back, top, bottom, left** and **right** views.

I'm working on point alignment for an upcoming release.

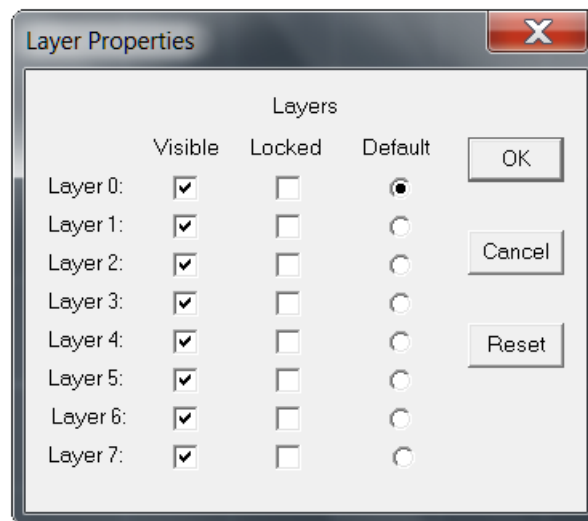


Alignment Ruler: Not aligned - Blue

Alignment Ruler: Aligned - Yellow

Layers

The Object, Figure, and Scene editors now support Layers. There are 8 editing layers where you can place each mesh, object, or bone in a specific layer. Everything is on layer 0 by default but this can be changed in **Layers Properties** dialogue box found under **Options**→**Layers** on the main menu. You can set the default layer to between 0 and 7.



Each layer can independently be displayed or hidden, and whether they are editable or locked. The top toolbar displays the layers' status. **0 1. 2** etc You toggle the visibility by clicking on a

layer. **Orange** numbers represent a visible layer that contains an element while **light gray** shows a hidden layer where your object is unseen in the viewport. **Dark gray** means that the layer is empty. Right-clicking locks and unlocks a layer, indicated by a small lock next to the number. Objects in locked layers are visible (if not hidden) but you cannot select or edit them.

In order to set an object to a specific layer just double-click on the object to open up its individual properties editor. Near the bottom you will find the new Editing Properties box.



Put the required layer number in the box to set its layer. The other properties will then become available as mentioned above. Don't worry if you accidentally put in a number higher than 7 because it will automatically default back to layer 7.

While layers are intended as a modeling tool they also have useful purposes in your Scene Mode. (Remember, you can already animate the visibility of elements of a scene using the Visible controller in the Element Editor under Other Animated Properties.)

Here's how the different Layers in the Object and Scene editor interact with each other:

In **Object Mode**: An Object has no single Layer number unless it is a single component. Individual components of an Object, however, can each have their own Layer. The properties of these Layers are set within the Object editor (Visible, Locked, etc.).

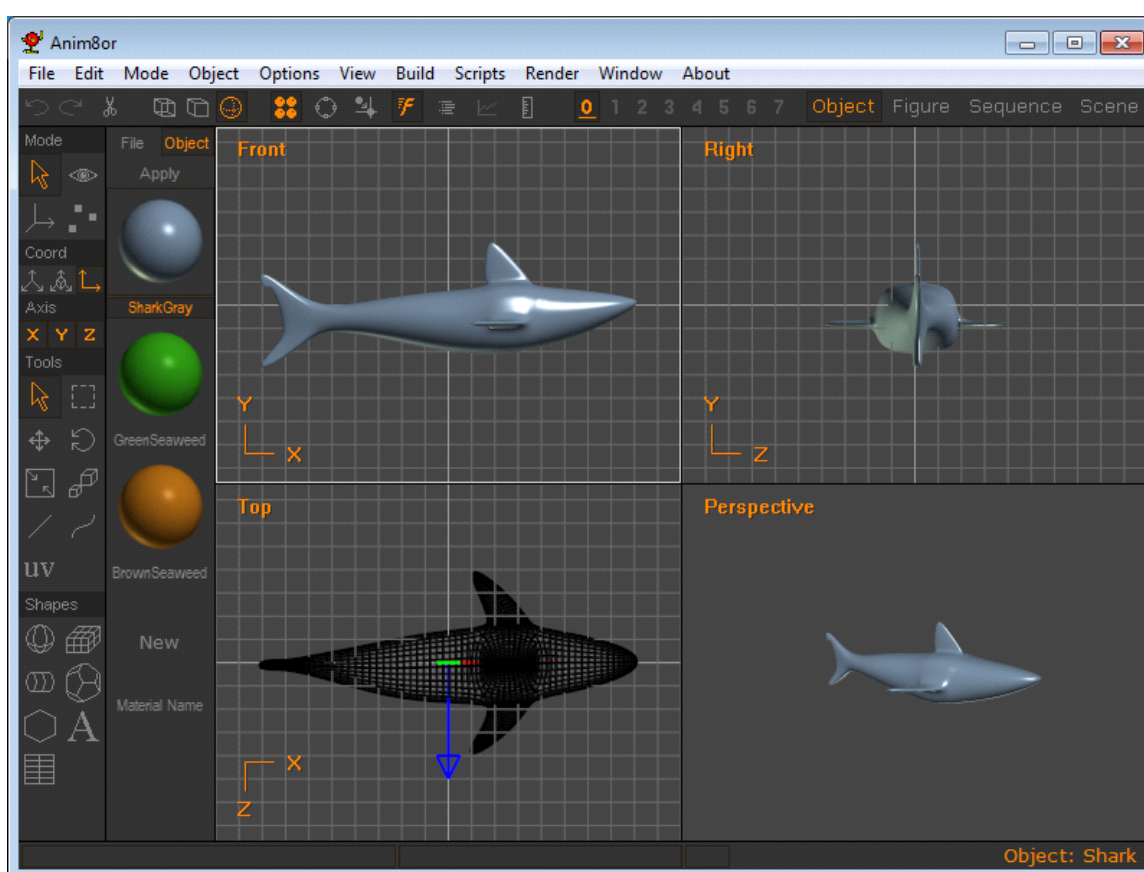
In **Scene Mode**: Each time you add an Object to a Scene, Anim8or creates a new Element to add to the Scene that references that Object. That Element has its own Layer which uses the Scene's Layer properties.

Layers can have many imaginative uses so be creative. :)

Object Editor Basics

The **object editor** is where you create and modify new 3D objects. You enter the object editor by default when you run Anim8or. You can enter it at any time by selecting **Mode→Object** in the menu, or by clicking on the Object tab to the right of the top toolbar.

The basic screen is shown below. On the left is a toolbar. It has icons that represent the most common operations that you will be doing. It allows you to change the mode that you are working in, add basic primitives, etc. You use the menu at the top of the screen to perform less commonly needed actions.



The object editor has four modes: **edit**, **viewpoint**, **axis**, and **point edit**. You change the mode you are currently working in by clicking one of the topmost four buttons on the toolbar. This will also change the lower part of the toolbar to display buttons appropriate for that mode.



Object/Edit mode button. This is the initial mode that the object editor starts in.



Object/Viewpoint mode allows you to pan, scale, rotate, and size-to-fit any or all of your views of your workspace.



Object/Axis mode lets you move an object's pivot or origin.



Object/Point Edit mode allows you to manipulate individual or groups of points, edges, and faces.

Object/Edit Tools

In Object/Edit mode there are some buttons just below the six small buttons on the toolbar that you use to manipulate components of your objects. Your actions only affect the selected parts of an object.



You can select something by clicking on it when you are in **select** mode. This button puts you in select mode. You can temporarily switch to select mode at any time by simultaneously pressing the **Ctrl** and **Shift** keys.



You can also select components by dragging a rectangle around them in **drag-select** mode.

These buttons allow you to:



move,



rotate,



scale non-uniformly, and



scale uniformly the selected components with your mouse.

You can also add and edit **splines** by adding



straight and



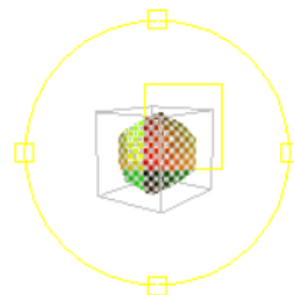
curved segments.

Splines are also sometimes called paths since they can be used as a path for motion.



texture coordinate editing tool. You use this tool to interactively apply texture coordinates to selected objects. It adds an overlay to the screen that is similar to the arc-rotate tool used for changing your viewpoint but for textures except that it's used for assigning texture coordinates to objects.

The yellow square represents a texture's basic size. You can move the texture around on your objects with the right mouse button, and you can rotate its orientation with then left button. The middle button scales textures.



Basic Objects

You can add new objects to Anim8or in two ways. The first is to import meshes for the **.3ds**, **.obj**, and **.stl** file formats. You use the **Object→Import** menu item to do this. The second way is to start with one or more built in **primitives** or shapes. The lower part of the Object/Edit toolbar has several icons for adding new primitives.



Sphere [S] - This button allows you add spheres to an object. First click on it. Then move the mouse to where you want the sphere to be added. Click and drag to create a sphere of the size you desire.

If you want to add several spheres, use the right mouse button. Using the left button returns to select mode after adding a single object.

Double clicking on the sphere brings up a dialog where you can change the sphere's properties from the default values.



Cube [C] - This button adds cubes and rectangular blocks. You can divide them into as many subdivisions in each axis as you like by double clicking on a block to view its properties dialog.



Cylinder [Y] - This button adds cylinders. You can taper them, leave the ends open or closed, and set the number of divisions used to make them in their properties dialog.



Platonic Solids [M] - Add various platonic solids and other built-in solids with this button. You can set the current solid type using the Build→Primitives command.



N-Gon [G] - This button adds unfilled polygons to an object. They are built as an editable spline so you can make a lot different shapes starting with the right polygon. Splines can be filled, extruded, lathed, edited, etc.

To set the number of sides, use the Build→Primitives→N-gon command.



Text [T] - You create true type text objects with this button. Just click somewhere in a window and start typing. If you want to change the properties, double click on the object. You can use any font and style.

Each letter is a single spline (possibly with multiple parts, like the hole in an O), and the whole string of characters is grouped together. You can extrude text objects directly using the Build→Extrude command.

You can also convert a text object into a group of general-purpose splines using the **Build→Convert-to-Spline** command.



Warp Modifier [W] - This is the modifier button. Use it to add modifiers, which can bend, stretch, warp, and twist another object into many new shapes. Modifiers are covered more fully in a later section of this chapter.

Each object in Anim8or has its own coordinate system. Its pivot is located at its origin. When you scale or rotate an object it is done using the pivot. When you are in wireframe mode, the pivots of selected objects are shown as red-green-blue axes like this:

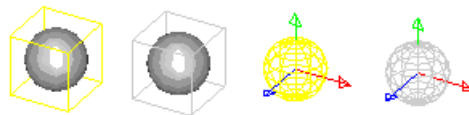


You can move and rotate pivots by using **Object/Axis** mode.

Mesh vs. Parametric Components

There are two kinds of components in an Anim8or object: **parametric** and **mesh**. Parametric components are defined by a small set of numeric values. These values are used to build a viewable mesh whenever they are displayed. You can edit these parameters by double clicking on a parametric component. Changing parameters will generate a new mesh for display. You cannot modify the mesh directly. Some parametric components are the sphere and cylinders described above.

Meshes however can be fully edited, smoothed, etc., even at the point or face level. You can convert parametric components into meshes with the **Build→Convert-to-Mesh** command. It is easy to see whether an object is a mesh or not by selecting it. In a filled view, selected parametric components are enclosed in **YELLOW** bounding box and selected meshes in **WHITE** ones. In a wireframe view, parametric components are shown in yellow lines and meshes are shown in white or some other color lines. The four spheres below show the same geometry. The first and third are in their original parametric form, while the second and fourth have been converted into meshes.



Once a parametric component has been converted into a mesh it cannot be reverted.

Forum Note

After working for a long time you may realize that your object wasn't centered in the bounding box and is now off center. Will this cause a problem later? Is there a fix? Often just selecting Join Solids is sufficient. Doing that will produce a new bounding box with the same orientation as Anim8or's axis.

Object Materials



You can view an object's materials by displaying the **material toolbar** with the **OPTIONS→MATERIALS** [**Ctrl-M**] command. Each named material is shown applied to a small sphere. Double clicking on the sphere brings up the material editor dialog allowing you to change that material's properties.

The selected material is shown as a depressed button with a white name. You can set the basic material for any selected component by clicking the **Apply** button.

The **Object** button shows the current object's materials. These materials can be used only by the current Object.

The **File** button shows the project's global materials, those that are shared among all objects in an entire project. You should add materials that are common to several different objects to the global list to help manage them easier.

If you want to define a new material, double click on the **New** button at the bottom of the list.

If there are more materials than can be shown, you can left click in the material toolbar and drag up or down to **scroll** the list.

Materials are discussed in more detail in **Chapter 9, Materials**.

Plug-in Shapes

You can add new parametric shapes to Anim8or beyond those that come as part of Anim8or. These are defined by special script files that you add to the Script Directory. They behave like ordinary parametric shapes and add a new button to the bottom of the left toolbar.



This is the default button for a shape plug-in if the script doesn't define one. You can set a plug-in shapes's parameters in the properties dialog by double clicking on it. You may also be able to scale the size or other parameters with the scale and scale non-uniformly tools, depending on how the plug-in is defined.





More often there will be a button definition as part of a shape plug-in. This one is for a spring. An example of a spring is shown below. The right spring has been converted to a subdivision shape.



See **Chapter 10 Scripts** for more information on adding plug-in shapes to Anim8or.

Splines

You use the curved path  and straight path  buttons to manipulate splines. To add a new spline, you must first deselect everything. Then select either the curved or straight path button and click and drag to place the first segment. The spline will be drawn with a white pick box at its root and a red one at its head like this:



The red box also indicates that the spline is extendable. You can left click on it and drag to add more segments:



Once you have built all of the segments you need, go back to select mode and double click on the spline to bring up its property dialog and uncheck the **extendable** check box. This prevents the accidental extension of your spline when you are trying to modify it in some other way. Your spline will look like this:



Now if you reselect either path button you can click on **knots** and display their control points:



You can drag the knots to new locations, and change the positions of the control points to alter the shape the spline. Double clicking on a knot will show the knot dialog where you can change the knot into either a **smooth** knot or a **corner** knot.

Double clicking on a segment will highlight that segment and let you set the number of straight lines used to draw it:



True Type Fonts

When you click on the text button you can add text to an object. Click in a view window and a **text** cursor will appear. Type some text and it will appear as an outline in the window. Double click on the text and the text dialog appears. You can change the **font**, make it **bold** or **italic**, and set its **size**. You can always resize a text object but this isn't always the same as changing its initial size with the font properties dialog. Larger fonts are sometimes generated with more detail.

Arial Times

You can also fill and extrude text. Use the **Build**→**Fill** and **Build**→**Extrude** commands:

Arial Arial

When they're filled or extruded, text objects are converted into a group of meshes. You can do anything to them that you can to an ordinary meshes.

Filling

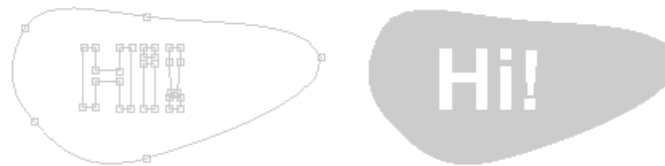
You can convert any closed spline into a flat, **filled** mesh. Just select that spline and apply the fill operator using the **Build**→**Fill** command. The spline should not cross over itself or it will not fill. To fill a spline, simply select it and the select the **Build**→**Fill** command.



You can also build complex splines consisting of multiple independent curves. Interior closed areas are filled as cutouts. To build complex splines, select several basic splines and apply the **Build→Join-Splines** command. For example:



Since text items are actually a group of (possibly) complex splines, text can also be combined with splines and filled. First select the text and convert it to splines with the **Build→Convert-to-Spline** command. Then ungroup the string of characters with **Build→Ungroup**. Finally select the spline along with all of the individual characters and combine them into a single multiple spline with **Build→Join-Splines**. Then fill the result:



Extrusion

Two more operations you can do on splines are **extrusion** and **lathing**. Both create a three-dimensional mesh out of a spline, but each in a different way. An extrusion sweeps the edges of the spline along a path (straight or curved) while lathing spins them around an axis.

To extrude a spline simply select it and choose the **Build→Extrude** command. This will display a dialog with several parameters that you can set, such as whether to cap the ends and which direction to extrude. One option is to extrude along another spline. When you choose this option, you will be prompted to select the spline to extrude along after you leave the dialog. Some extrusions are shown below:



The top left "Y" shaped spline is shown extruded along the Z-axis, and along the lower sine-wave shaped spline.

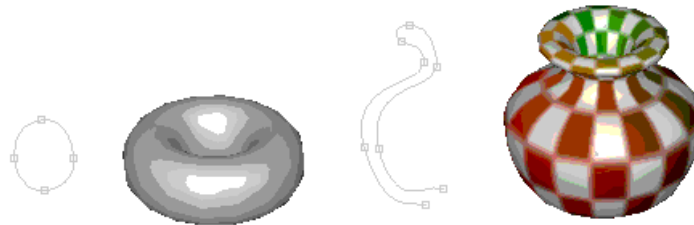
Note: The "center point" for the extrusion is the initial origin of your spline. Make sure that you create the spline around the origin or the results may not be what you expect.

You can also extrude text. Just select it and apply **Build→Extrude**:



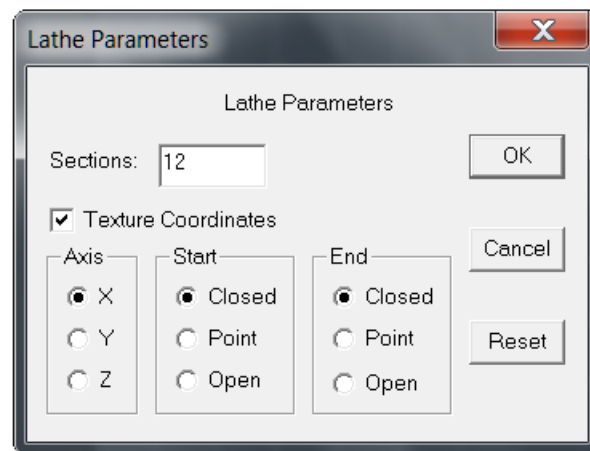
Lathing

Lathing creates a solid surface from a spline by spinning it around an axis and converting the area it sweeps into a mesh surface. It can produce a variety of different objects depending on the shape of the spline that you lathe. If you lathe a closed spline you can create wheels and donuts. If you lathe an open spline, you can build wine glasses, pots, and vases:

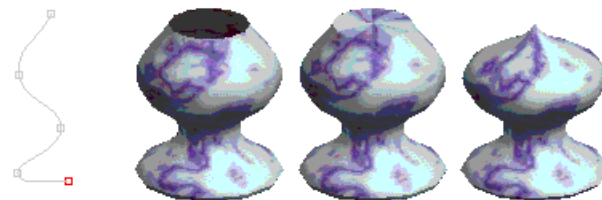


Since you can apply texture coordinates at the same time that you are lathing a spline, it is easy to build things with detailed colorings.

You use the **Build**→**Lathe** command to lathe a spline. This displays the dialog:



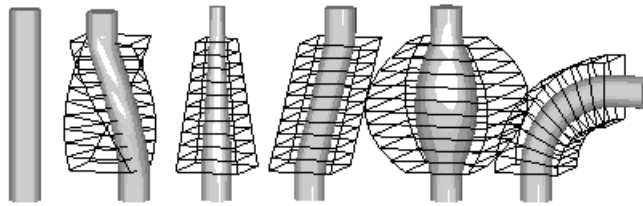
The *Axis* section selects which axis will be used to do the rotation. You can control the behavior of the end points of an open spline with the settings in the *Start* and *End* sections. The three objects below are all lathed from the same spline, but the starting point (the one at the top, not the red one) is handled differently in each case. **Open** leaves the points at the beginning and end in their original place. **Closed** adds a new point on the axis at the same level as the end point and connects to all the end points creating a flat top or bottom. **Point** moves the spline's end point precisely to the axis and then lathes the spline.



Open, Closed, and Point modes used at the Start

Modifiers

Modifiers allow you to stretch, bend, twist, and warp meshes into an endless variety of alternate shapes. To use them, create a modifier object and **bind** it to an existing mesh with the **Build→Modifiers→Bind** command. You can double click on the modifier to bring up its property dialog and change its behavior. The effect it has on the target mesh is shown in your object views. When you are satisfied with the shape, select the **Build→Modifiers→Effect** command and the mesh will be transformed into its new shape.



Here is an example of a cylinder modified by a **twist**, **taper**, **skew**, **swell**, and a **bend** modifier. Modifiers preserve texture coordinates and other properties bound to points on the object.

Mirror Image

You can also make a **mirror image** of an object using the **Build→Mirror-Image** command. This builds a mirror image of the original object, a duplicate with the points reflected to the opposite side of the X, Y, or X-axis. These hands are simple mirrors of each other:



If you are building a symmetric object you can save time by building half of it, mirroring that half, and joining the two together.

Smoothing

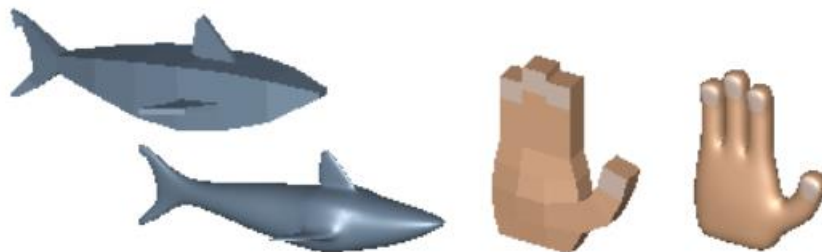
You can use **smoothing** to round the corners of basic shapes and to give more detail to the model. Anim8or can subdivide any mesh. Just select the objects that you want subdivided, and select **Build→Smooth-Object** from the menu. Each time you do this you will round the corners a bit more, and multiply the number of facets by four. So be careful, or you'll create a very large model to manipulate!



***Note:** Smoothing used to be called **Subdivide Faces** in earlier versions of Anim8or.*

There is a modeling technique called **box modeling** that you can use to model complex objects that begins with a simple box. You bevel, extrude, resize, and warp a few flat faces until you

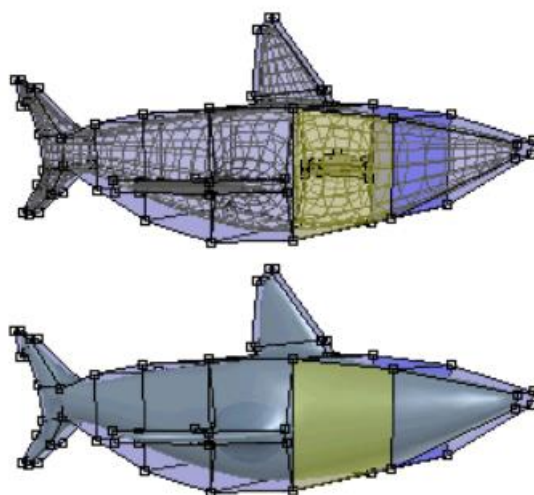
make a rough form of your desired object. Then you apply subdivision, and, viola! You have a nice model. The following were made in Anim8or using box modeling:



Not a bad hand considering it was modeled by 24 simple cubes! You can learn more about building the box shapes in **Chapter 4: Object/Point Editor**.

Subdivision Objects

You can convert a Mesh into a **subdivision object** with the **Build→Convert-to-Subdivided** command. This allows you to edit Subdivision objects just like regular meshes, and they are automatically subdivided as you work. This is often an easier way to model than to continually apply subdivision operations to try and see what your model will eventually look like. Selected faces are shown as transparent windows revealing a further refined model inside:



When you resize or extrude a face the internal smoothed view changes shape to follow. (You only see this view in the **Point Editor**. All other places your model will be shown as fully subdivided images.)

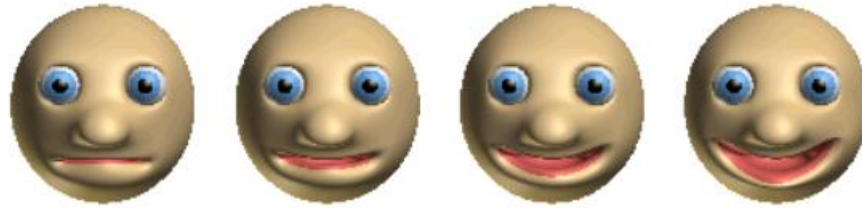
You can convert a Subdivision object into a Mesh with the **Build→Convert-to-Mesh** command. It will be converted at the subdivision level that you are using to view it in the working views.

Another very important advantage to Subdivision objects is size. Subdivision objects are stored in the undivided format that is many times smaller than the final fully divided model. You can control the level of subdivision used on the screen and in the final image by double clicking and editing the properties dialog.

Morph Targets

A **morph target** is a deformed version of an object where some or all of its vertices are moved to alter the shape. For example you may have a basic model of a character's head with a neutral

expression. If you want to animate your character opening and closing its mouth you'd create a morph target with the character's mouth open. You can then smoothly animate between the base shape with a closed mouth and the morph target with an open mouth using a single controller. As you increase the value of the controller the character's mouth opens.



The face on the left has a neutral expression. The face on the right is a morph target created for the face so that it's smiling. The faces in-between are morphs with a value of 1/3 and 2/3.

To create a morph target, you first build a base model. It's important to put the finishing touches on your model first since altering the base model can corrupt any morph targets you have added. You then make a morph using the **Build→Morph-Targets→New** command. Make sure to give it a meaningful name such as "smile" so you can find it when you are animating in the Scene editor.

Next edit your model in the Point Editor by moving vertices to alter the shape. When you're finished with the morph use the **Build→Morph-Targets→None** command. to return to the base shape or continue to add more morphs.

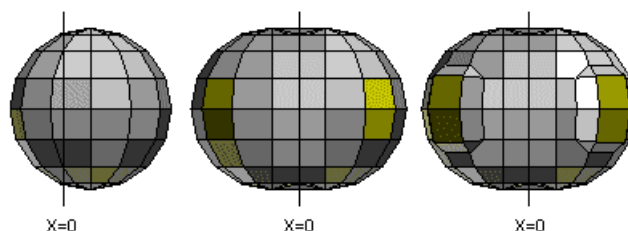
***Note:** If you use a Subdivision object for your model you will have fewer points to manipulate.*

You're now ready to animate your object in the Scene editor. Morphs show up in Scene Mode in the timeline, not in sequence or figure mode. Your list of morphs will show under the figure's name and can be applied to give expression to the face anywhere in the sequence (in scene mode) using key values of 0-1. To activate the morph or morphs you select the object with the morphs attached then use key 1 to activate or 0 to deactivate in the actual timeline.

Continuously Mirrored Meshes

Mirroring is a mesh property that helps you build symmetrical models. This property causes any change to the right side of a mesh to be applied identically to the left, and visa-versa. You can apply mirroring to any mesh with the **Build→Mirroring→Convert-to-Mirrored** command.

When you initially convert a mesh into a mirrored mesh, only points to the right of the Y-Z plane are mirrored, those with positive X coordinates. All points to the left of the Y-Z plane are deleted.



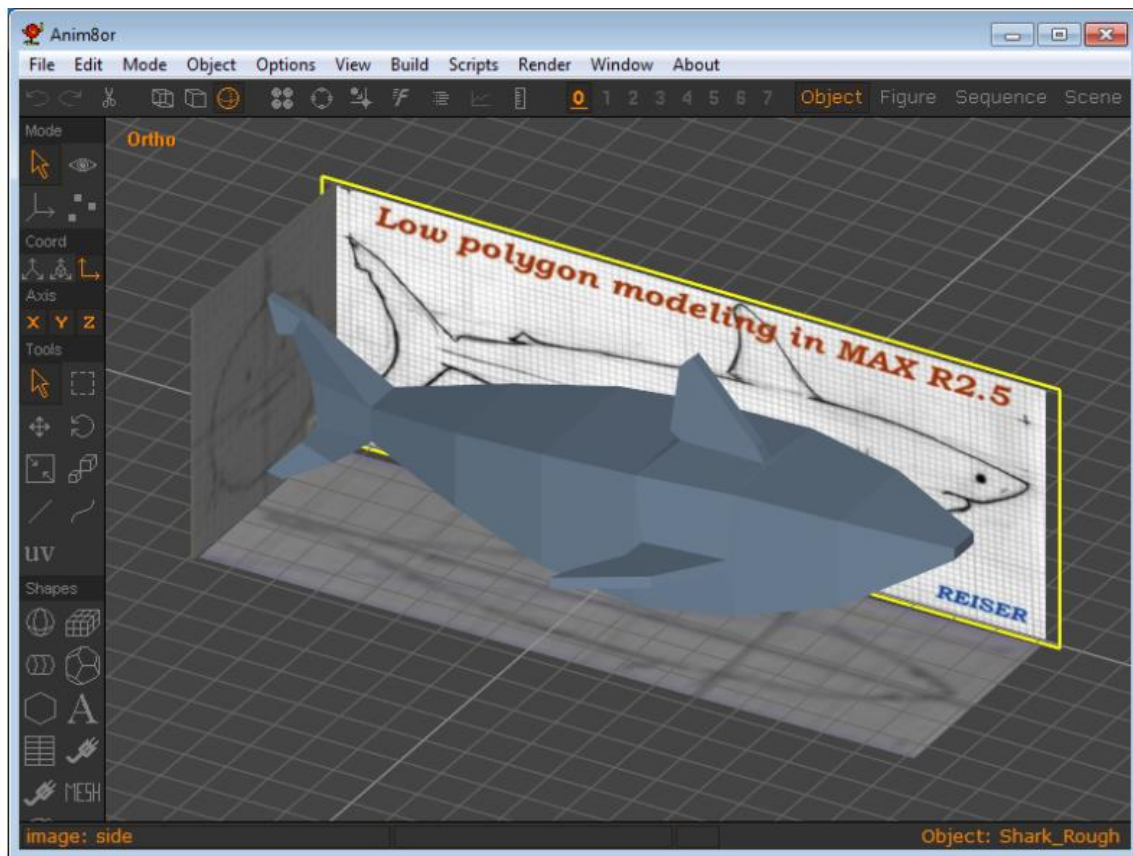
Once mirroring is enabled you can move, rotate and scale your model like any other model, without the need to keep it aligned to the Y-Z plane.

When you are finished with your model, or if you want to start making changes to only one side, you can disable mirroring with the **Build→Mirroring→Stop-Mirroring** command. If you re-enable mirroring you will lose any asymmetrical aspects. It's a good idea to save a copy of your mesh at the end of the mirroring phase in the event that you want to refine the mirrored aspects.


Note: At this time continuously mirrored meshes do not work with Morph Targets. You first must disable mirroring to start adding morph targets to your models.

Reference Image

You can add **Reference Images** to an object to use as a guide when modeling with the **Build→ReferenceCommand** command. References are added parallel to the current view, so if you are in the front, top and left views you can model in all 3 dimensions directly over the images. You may need to move them deeper behind your model so that they aren't blocking the view of your model.



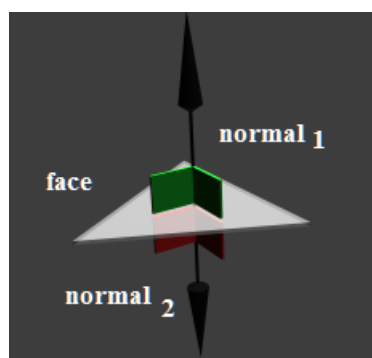
Object Point Editor

The Object/Point Edit button  [P] on the toolbar changes Anim8or into Object/Point editing mode. Here you can modify individual points, edges and faces of any mesh object. You can move, scale, rotate, add and delete them. You can extrude, twist and scale faces, weld close points together, etc. When you enter **Object/Point Edit** mode you will usually want to change the display into wireframe with the **View→Wire** [Ctrl-W] command.

Points / Edges / Faces

In 3D modeling the basic object that is used is the **point** or is more commonly known as the vertex. This is simply a physical representation of a three dimensional coordinate in space. Two points connected by a straight line is called an **edge**. When you have three points and they are all connected by edges you have the most basic polygon that can be made, the **triangle**. Since flat planes are determined by three separate points in 3 dimensional space this is the smallest polygon that can be used to create a **face** (the area within the edge boundaries). Each face is composed of two parts, the frontface (outward) and the backface (inward). Their orientation is important when assigning materials and for rendering (see Normals below). Polygons can consist of more than three points/edges but they may not necessarily be flat. Another common polygon used in 3D modeling is the **quad** or a polygon with four points/four edges. By combining polygons together using the same points you create what is termed an **element** (object). Everything that you can add to a scene, including lights, objects, figures, and targets, are also called **elements**.

Normals




Surface Normals





Normals are basically three dimensional orientation vectors of the faces of your polygons that are perpendicular to the flat plane of the face. These vectors are then used in rendering to calculate how light will react to the surface whether for materials, reflections, refractions, and shading. On curved surfaces the normals are determined by the normal of the tangent plane of the curved surface at the same position. Both of these normals are the same. In some lighting cases normals are determined by the orientation of the surfaces' vertices.

Unless specified, usually only the frontside of the face is determined and if the face normals are inverted then it would appear to be missing. It is always a good idea to check whether your normals are correct after modifications to your object or after importing a model from another


source. Quite often the normals are flipped and this can create problems if one does not correct them. Some programs do allow for the use of both sides and this should always be kept in mind when transferring objects between applications.


Object/Point Operations

Just below the coordinate system button row there is a new row of three buttons: . You use these to change the selection mode to **point**, **edge** or **face** in order to select different parts of your model. Left-click on these to select just one mode. Right click to toggle individual buttons.




The select  and drag select  modes will then select the corresponding items. There are two buttons that you will find helpful when selecting points, edges and faces. They control whether you can select **front facing**  [F] and **back facing**  [B] faces, edges and points. By default both kinds are enabled. Click on the buttons to toggle the setting.

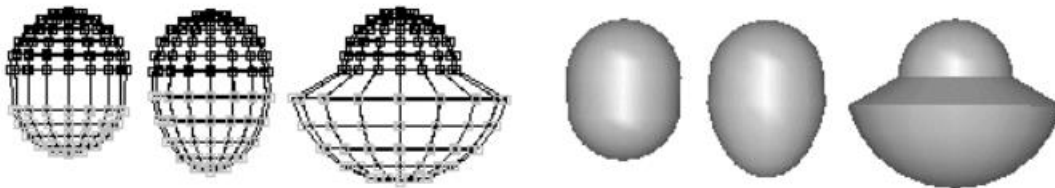
Point Editing

You use **point select** mode  [P] to modify the shape of an object, and to add and delete unwanted parts of an object. You can manipulate selected points by using any of the move, rotate, and scale buttons. The scale and rotate buttons operate of groups of connected edges, scaling and rotating each group around its center.

If you delete points using the scissors button  [Ctrl-X], or the delete key [Del], any edges and faces that contain those points are also deleted. For example, to delete the bottom of a sphere, first select all of the points below but not including the equator using drag select mode, and delete them




If you move  these same points, scale them non-uniformly , or scale them , you can make objects like these:



You can use the *Arrow Keys* on the keyboard to make tiny moves, rotations, and scales to selected points as well when the corresponding toolbar button is enabled.

In move mode, click-dragging with the middle mouse button (or alt-right) will move individual points along their normals instead of relative to the screen.

Edge Editing

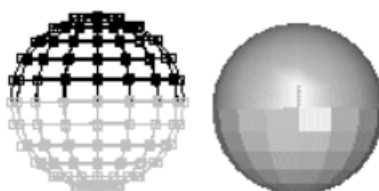
You use **edge select** [E] mode  in a similar manner to point select mode. When you select an edge, both of its end points are also selected. Thus all operations from point select mode are enabled as described above.

You can move, rotate and scale selected edges with the normal buttons. The scale and rotate buttons operate on groups of connected edges, scaling and rotating each group around its center.

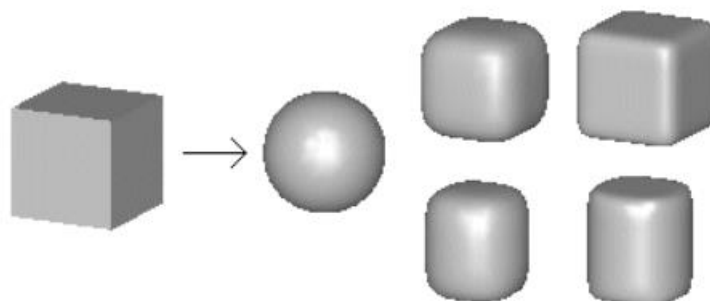
If you delete using the scissors in edge mode, however, only the selected edges and any faces that contain them are deleted. All points are saved. This allows you to change the tessellation of an object more easily, without having to add back previously deleted points.

You can also set various **edge properties** for selected edges. Select the edges you want to effect, and then select the Edit→Edge-Properties command. The default setting for edges is **smooth**. This means that if the solid angle defined by the two faces is less than a certain threshold, the edge will be smooth. Otherwise it will be creased. You can change the value of this angle by double clicking on a mesh when in Object/Edit mode to bring up its parameter dialog.

You can mark an edge as **creased**, so that it won't be drawn as a smooth surface but will appear as a sharp crease. This does not change the geometry of the model in any way; just the way normals are computed so that it appears smooth or creased. The following sphere has the lower half of its edges set to creased:



Another property that you can set in this dialog is how **rounded** an edge will be after it has been smoothed. Normally when a mesh is smoothed, all edges become smoother. Each application of smoothing increases the number of faces in a mesh, and smooths all edges. You can mark any edge so that it will not be smoothed, but will retain a sharp crease, for one or more cycles of smoothing. The following shows a cube and several possible smoothings of it:



The sphere-like ball is the result of a simple cube being smoothed several times. The top two rounded cubes are the result of smoothing a cube with all of its edges set to a rounded level of 1 and 2. The bottom two rounded cylinders are from a cube with only the top 4 and bottom 4 edges set to a rounding level of 1 and 2.


Forum Note: Edge-Point Joint selection

Points selected when edge selected:

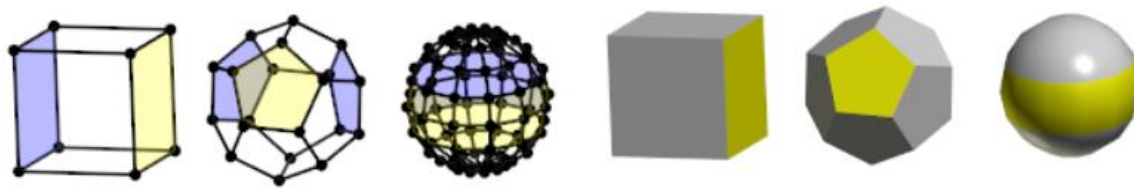
"That came in handy for locating a hidden point. "

Sometimes you need to move a point to move the edge attached. When the point is hidden by a cluster of other points, the easiest way to locate the correct point is by selecting the edge and following it along visually until you can see the point at the end of it.


Face Editing

You use **face select** [g] mode  to select and manipulate the faces of a mesh. You can delete selected faces with the scissors button or delete key, but points and edges remain unchanged. You can move, rotate and scale selected faces with the normal buttons. The scale and rotate buttons operate of groups of connected faces, scaling and rotating each group around its center

Selected faces are shown as filled even in wire frame mode. The front side is shaded **YELLOW** and the backside **BLUE** to help you see what faces are selected. This is also the case in filled mode, but you usually only see the yellow front sides.

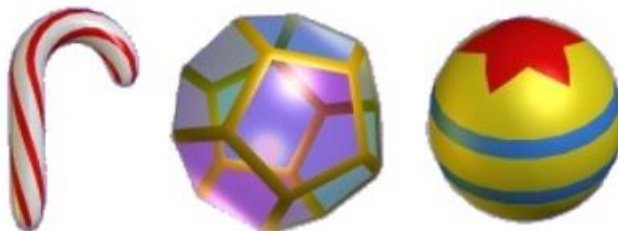


Applying Multiple Materials

You can also apply different material properties to individual faces. Simply select the faces that you want to change, display the material toolbar (use **Options**→**Materials** [Ctrl-M] if it isn't already visible), select the material to apply, and click the apply button  in the toolbar.



You can use as many materials on a mesh as you want. With a little imagination you can make objects like these:



The UV texture tool is also available in the Point Editor. You first select the faces that you want to texture, and then use the UV tool to interactively apply the coordinates. It helps if you already have a textured material applied to those faces.


It does not matter what size the actual texture image is, the UV coordinates specify a relative position in the image, what percent across or up and down, to apply to a face.


You can resize the texture using the UV tool in the Point editor.

1. First select only the faces with the texture you want to rescale.
2. Then click on the UV button and
3. click Yes in the dialog to enable texture UV generation for the object.
4. Click drag in the middle of the big yellow circle on the screen with the MIDDLE mouse button to rescale the texture. Use the RIGHT mouse button to move it if it's off center.


A Note on Selecting Faces

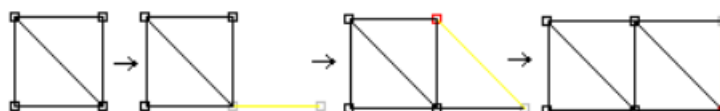
It can sometimes be rather tedious to select the exact faces that you want, especially in a complex model. There are several tricks that can help out.

The **select** mode  selects faces that are facing the viewer over those facing the other way. Often changing the view of an object, or rotating the view with the arc-rotate tool, so that different faces are in front, can help. This is one reason that selected faces are shown in **YELLOW** even in normal views where you normally can't modify individual faces.

The **drag select** mode  only selects faces that are entirely within the selection rectangle. You can often add faces one-by-one until you've selected what you need. The front/back buttons can be useful here as well.

Adding Points and Edges


You can add new points and edges to a model with the Add Edge  button in the toolbar. To add edges to an existing object, click on one of its points and drag the mouse out. A new edge and point will be added where you release the mouse. If you move the mouse to an existing point the end of the edge will snap to the existing point, thus creating an edge between the two points. You can build complex meshes quickly using this method:



If you want to add a point that is close to another one but not make it the same point, hold down the *Shift* key *after* you click the mouse button down and hold it until you release the mouse.

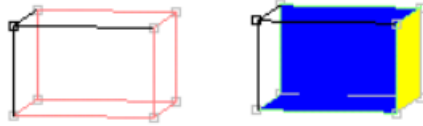
If you want to start a **new** mesh, press the *Shift* key *before* you press the mouse button. A new mesh will be created with a single edge and 2 points.

Note: This command only adds Points and Edges, not Faces. You can use the **Edit→Fill-Holes [J]** command described next to add new faces.

If you enable **grid snap**  [*Ctrl-G*] when you add points you can make objects with exact measurements more easily.

Adding Faces

Once you have added new edges to a mesh, you will be able to fill in the gaps with new faces. You do this by first selecting all of the edges that surround where your new face or faces will go. Then you use the **Edit→Fill-Holes** command to fill them.



There are a few things to note before you start adding faces:

- Faces can only be added to completely connected loops of edges. If there are any gaps, no face will be added.
- All edges in the loop must *not* already be part of two (or more) faces. Only edges that are part of *at most one face* will be considered. You can tell how many edges an edge participates in by its color when selected:

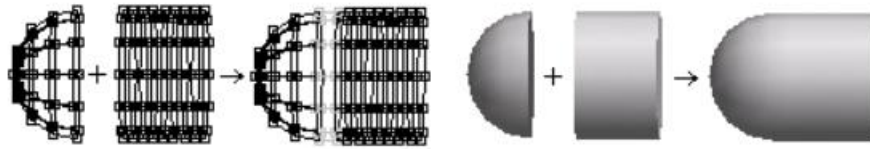
Color of Edge	Number of Faces
RED	None
GREEN	One
WHITE	Two
VIOLET	Three or More

- You can select multiple edges and do a fill of the whole batch at once. Anim8or will start by filling any selected edge loops of length 3. Then it will fill those of length 4, etc. until no more faces can be added. This will sometimes create undesirable results and you will have to try again in smaller batches.
- You may have to apply the **Edit→Flip-Normals** [N] and the **Edit→Fix-Normals** [X] commands to orient new faces properly after they are added.
- You may have to delete some faces in your mesh before you can add new ones.

Connecting Meshes

You can attach two separate meshes to create a single, connected mesh using the JoinSolids, Add Points/Edges and Add Faces commands.

- First position the two meshes with the parts that you want to attach near each other. Leave a small gap for a row of new faces that will be used to connect them.
- In the Object Editor, select both meshes and apply the **Build→Join-Solids** command. This doesn't add any new connecting faces but combines them into a single Mesh allowing you to do so.
- Next in the Point Editor, add new edges between the pairs of points that you want to attach.
- Select all of the edges that you just added, and those on the new boundary.
- Use the **Edit→Fill-Holes** [J] command to add the connecting faces.



Merging Points

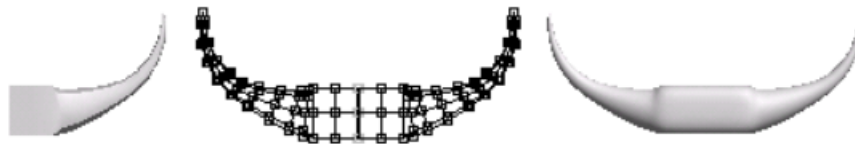
You can merge separate points into a single point with the **Edit→Merge-Points [L]** command. First select the points that you want merged. It is best if they are very close to each other to begin with. Then apply the **Edit→Merge-Points** command. A dialog appears where you can enter the maximum distance allowed between pairs of points. Click OK and all eligible pairs will be merged.

Connecting Meshes (2) by Merging Points


You can also use this technique to connect two meshes. It works best if the connecting point sets are similar in size and number. For example connecting half of a Sphere that has 12 longitude lines with a cylinder with 12 sections around is a snap.

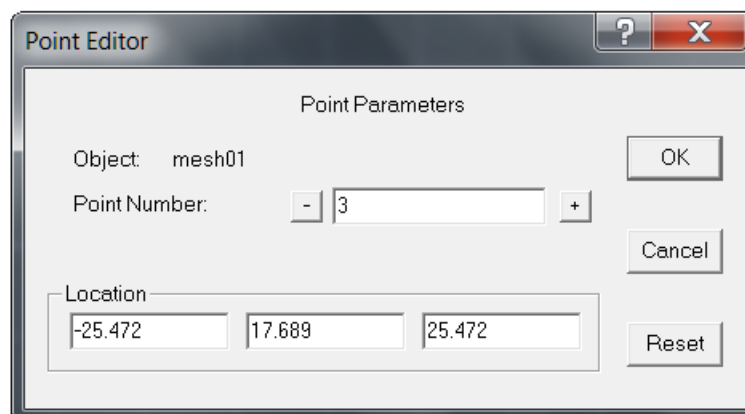
Connecting Meshes (3) by Merging Points

You can often build one half of a symmetrical object, then mirror it with **Build→Mirror** and join the two halves into a single mesh with **Build→Join-Solids**. Then position the two halves close together, select the adjacent points and use **Edit→Merge-Points** in the Point Editor to make a seamless connection. This merged object was then smoothed one time to round off the corners.



Point and Line Parameters

When you are in Point Select  mode you can double click on a point to bring up a dialog box that lets you read and alter its position numerically. This is very useful when you want to precisely position a part of a mesh when building a mechanical model.



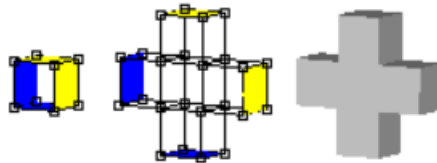
When you are in Edge Select  or Face Select  mode you can similarly double click on an edge to edit its properties.

Face Extrusion and Manipulation Tools

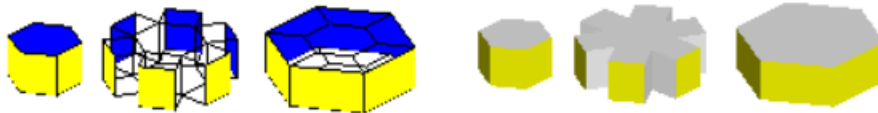
You can manipulate selected faces in a variety of ways using the lower group of buttons in the toolbar. They can be used to build very complex models from a small number of primitive shapes using the box modeling method.



Extrude -The **extrude** [*X*] tool extrudes all of the selected faces. Each face is translated in the direction that it faces, either out or in, and new polygons are added to connect the face to its original location.



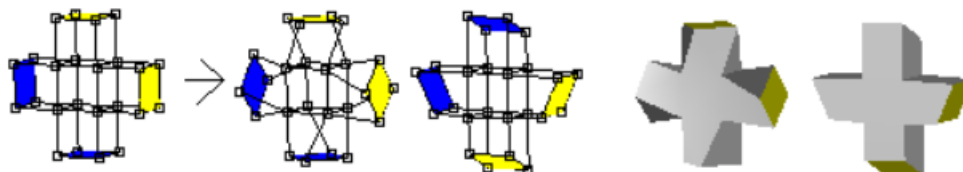
Extrude-connected -Similar to the extrude tool, the **extrude-connected** [*Y*] tool extrudes selected faces in a Mesh. However adjacent selected faces remain connected. New faces are only added to the edges between a selected and an unselected face.



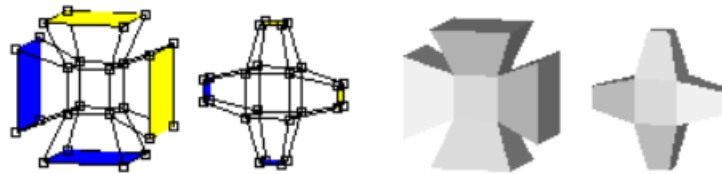
In the image above the 6 faces on the first solid are shown extruded and extrude-connected.



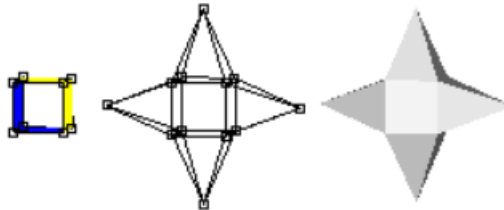
Rotate -The usual rotate button **rotates** [*R*] each selected face around its center. If a group of adjacent faces are selected the entire group is rotated around the group's center.



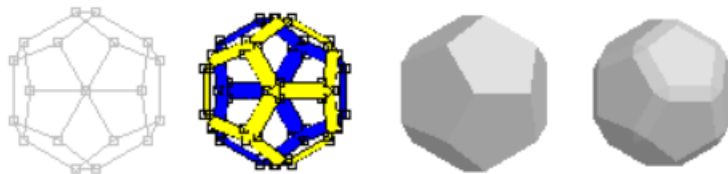
Scale -You can **scale** faces both **uniformly** [*s*] and **non-uniformly** [*n*] using the usual scale buttons. As above, adjacent selected faces are scaled as a group.



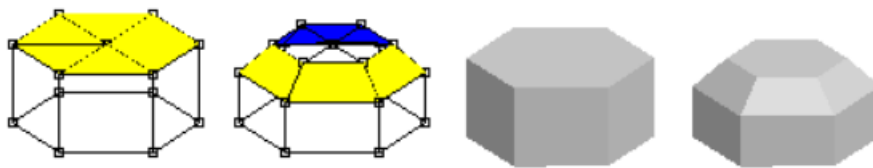
Peak -This one will replace faces with a **peak** [*P*].



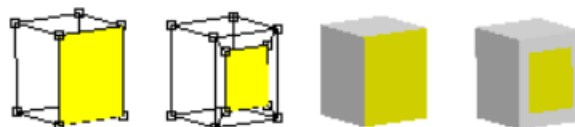
Bevel -The **bevel** [*b*] tool is useful for adding beveled corners to surfaces. You can bevel faces, edges and points of any mesh. Selected edges and points are turned into new faces, cutting the corner off of the original model as shown below:



Selected faces are handled as a group. Edges between a selected face and an unselected face are beveled but edges between two adjacent selected faces are left alone.

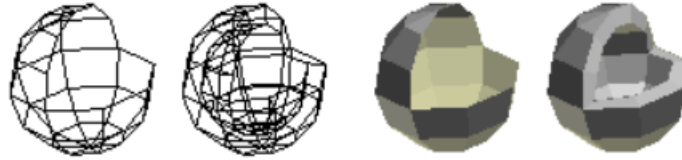


Inset -The **inset** [*I*] tool is also useful for adding faces to surfaces. Each selected face is replaced by a smaller version of itself and new faces that connect it to the original edges. This is an easy way to add more detail to a particular part of a model, or to make a hole for a door, window, or eyes.

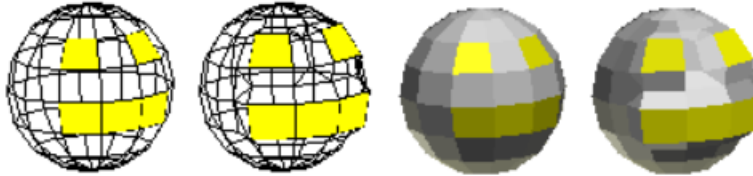




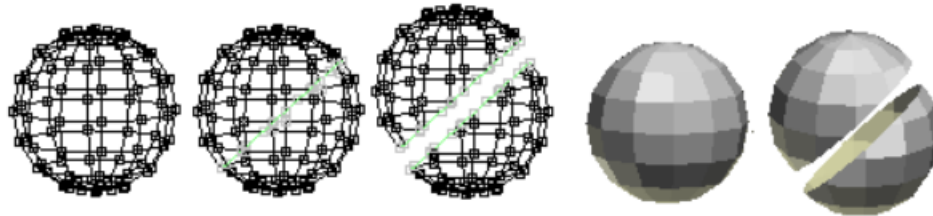
Shell -The **shell** [**u**] tool adds thickness to the walls of a Mesh. To make a shell simply click on a Mesh or a Subdivision shape and drag the mouse. The basic Mesh cannot be completely closed. It must have at least one "missing" face.




Bump -The **bump** [**B**] tool raises all the selected faces up on a bump.



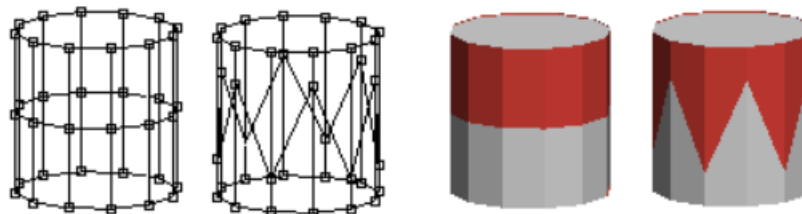
Cut Faces -If you want to cut a slice through a model you'll find the **cut faces** [**C**] tool quite handy. It slices through edges and faces when you click-drag the mouse across an object. If you slice all the way across an object as shown below you can then use the Edit→Loop-Cut command.



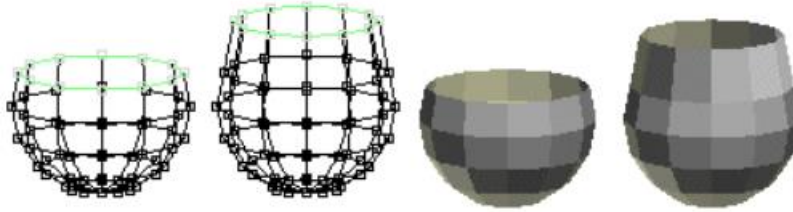
You can divide a face into two by clicking on one vertex and dragging to another. The connecting line will snap to a vertex when you click nearby. Or you can disable the snapping if you press and hold the **Shift** key when applying a cut. You may want to disable the back facing edges from being cut as well by turning off the  button.



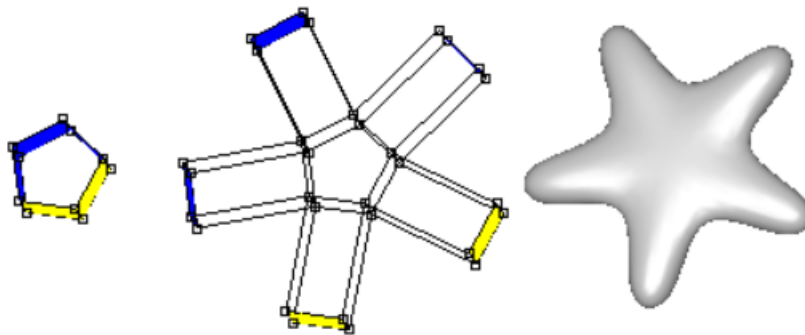
Slide -The **slide** [**S**] tool lets you slide a point along an edge, or extend the edge. Just click on a point and drag the mouse in the direction of one of the edges that enters it. This will select the active edge and shorten it. If you want to lengthen it instead just keep dragging but in the opposite direction.



Edge Extrude -The **edge extrude** [**E**] tool extends your object for all selected edges that have only one face, adding a new adjacent face.



When you use these face manipulation operations you may wish to convert your mesh into a **subdivision mesh** with the **Build→Convert-to-Subdivided** command in order to build more complex shapes. For example, start with an extruded pentagon, select all 5 outer faces, extrude them, and convert to subdivided to make a starfish:



Alternately you can use one or more applications of the **Build→Smooth-Object** command for the same effect.

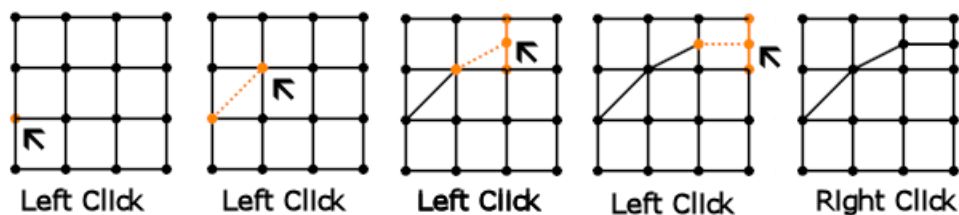
Topographical Knife



The **topographical knife** or topo [*k*] tool is a major addition to version 1.0. This is an extremely versatile tool that allow for quick editing of meshes in multiple ways.

- **Split Faces**

LEFT-click on a point or edge begins adding edges to split faces giving them more detail. Release the button and move to another edge or point on the same face. Click again to divide the face in to two. You can continue adding a sequence of connected edges in this manner. RIGHT-click to exit this mode.



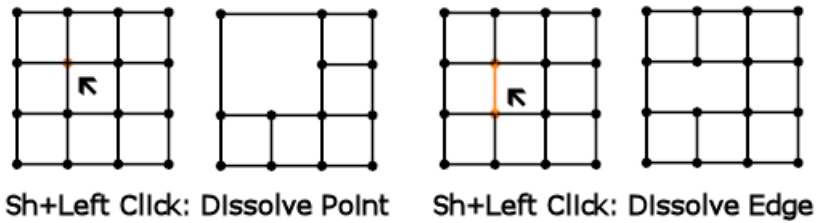
- **Add Edges**

Ctrl + LEFT-click allows you to add edges arbitrarily, not just across a single face. This includes adding edges connecting across gaps and isolated points not in any face. As long as you keep pressing **Ctrl** you can add multiple, arbitrary edges. Releasing **Ctrl** reverts to the original behavior. Again use RIGHT-click to exit.

Anim8or V 1.0 build 1.01.1318 Manual

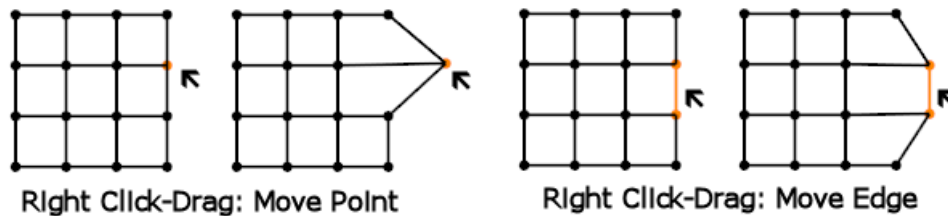
- **Dissolve**

Shift + **LEFT**-click on a point or edge to remove or "dissolve" it. Your mesh will reform and stay connected in a simpler form. This is a useful way to clean up areas of your model with too much detail or awkward geometry.



- **Move Edges and Points**

RIGHT-click on a point or an edge and **DRAG** to move that point or edge along with the mouse. Anim8or uses the active coordinate system for the move, so make sure you've selected the one you want.



- **Move by Normal**

Shift + **RIGHT**-click and **DRAG** moves in the direction of the point's normal.

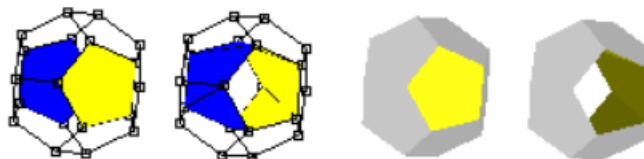
Face Editing Commands

Anim8or has several other useful face editing commands.

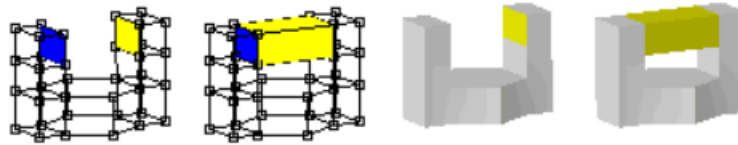
You will find them under the Edit menu in the Point Editor.

- **Bridge**

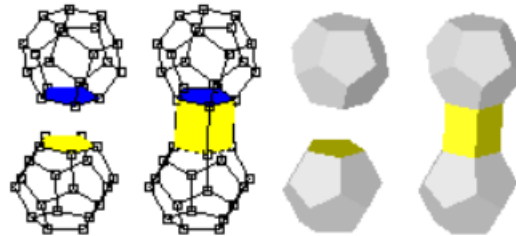
The **bridge** [=] command allows you to do several useful operations. To use it you first select two different faces. They have to have the same number of edges but can belong to different Meshes if you want. The **Edit**→**Bridge** command will then connect the two selected faces with new faces. You can poke holes through a Mesh:



You can connect two parts of a Mesh to form a bridge:

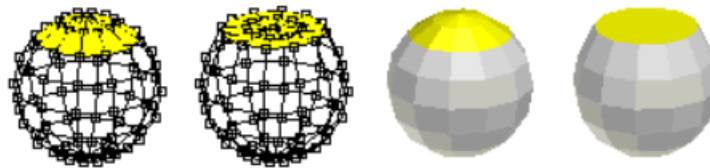


And you can join two different Meshes into a single mesh:



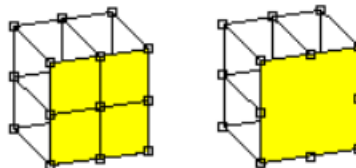
- **Flatten**

The **flatten** command does just what its name suggests: it flattens groups of selected faces. All the vertices of each group are projected onto a flat plain. You can choose the group's average face direction or the X, Y, or Z-axis. Flattening is quite useful when you need a level base for a model, and can be used with the merge face command to help simplify your models.

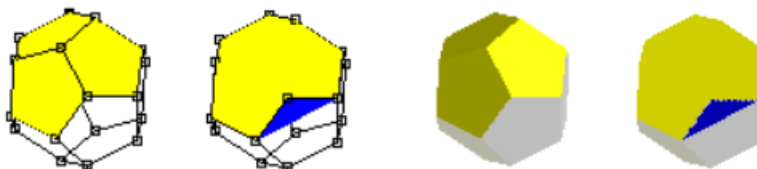


- **Merge Faces**

The **merge faces** [*M*] command combines groups of selected faces into a single face. You can simplify areas of your models that are too detailed this way.



You have to be careful to keep the faces **convex** and relatively flat or they may not render properly. Convex faces are those that don't have any points inside their outer boundary. Faces with interior points are called **concave** and can cause rendering errors:



In the example shown above three pentagonal faces are merged into one big face with 9 sides. It is concave and doesn't draw properly as a result. It's easy to see in this example because of the stray back facing part that is blue when selected. Other times it may not be as obvious what is wrong.

If you are working with a Subdivision object this is not as important. The faces that you are editing aren't drawn in your final object. Subdivision will smooth the mesh making it much better formed along the way.

Miscellaneous Commands

There are several miscellaneous commands in the Point Editor that you will find useful. One group helps you make certain useful of selections that could otherwise be quite tedious. The other group modifies objects in various ways.

Selection Commands

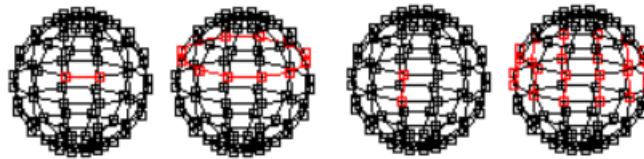
You will find these commands under the **Edit**→**Select** menu.

- **Quad Loop Select**

A **quad loop** is a chain of adjacent quads, or 4 sided faces, that are attached across opposite sides. The **quad loop select** [*q*] command extends selected edges along quad loops as long as there are exactly four faces joined together at each vertex. For models like the sphere below this results in a closed ring around a latitude line. For vertical edges it only extends to the poles since there are more than four faces sharing the pole points.

- **Quad Ring Select**

The **quad ring select** [*Q*] command works on chains of adjacent quads, or 4 sided faces. When a quad has a selected edge, the opposite edge is also selected. This is repeated until all quads in a "ring" are selected, or until a face without 4 faces is encountered. (Selections are shown in red here to make them better visible against a white background.)



Editing Commands

These commands are under the **Edit** menu.

- **Connect Edges** [*j*]

adds new edges between the centers of any selected edges that share a face.

- **Cut Edges** [*Z*]

splits each selected edge into two in the middle.

- **Detach Faces** [*D*]

removes all selected faces from their mesh and adds them to a new mesh.

- **Subdivide Faces** [*U*]

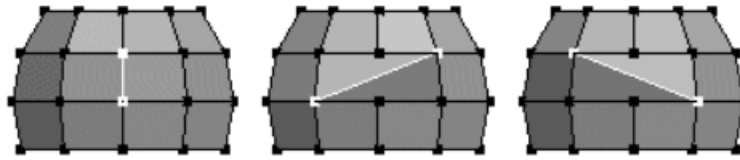
divides those faces into several smaller faces. It does this by adding a new point in the center of each selected face and connecting it to the center of the edges of that face.

- **Loop Cut [L]**

splits a mesh into multiple parts along closed loops of selected edges.

- **Spin Quads [V]**

Spin quads is a useful tool for modeling. It changes which corners of a face are connected by rotating an edge to adjacent points. The faces adjacent to the selected edges must have 4 sides.



These commands are under the **Edit**→**Select** menu.

- **Invert Selection [i]**

selects all unselected points, edges and faces and de-selects those currently selected.

- **Grow Face Selection [G]**

selects any unselected faces that are adjacent to selected ones.

- **Select Connected [c]**

selects all points that are in a connected region of the mesh to at least one selected point, all edges that are in a connected region to selected edges, and all faces connected to a selected face.

- **Select by Material**

allows you to select all faces with a given material.

- **Select Malformed Edges**

selects any edges that are in 3 or more faces. These edges can have unpredictable side-effects when you edit or render.

- **Select Orphans**

selects any points and edges that are not currently being used in a face. This can help you remove unnecessary geometry from your models.



Fast Select

Fast Select allows you to select and model on the fly without changing to the Select tool. This can be disabled for beginners who sometimes find it easier to separate these functions. Enabling **fast-select** allows a more fluid selection and editing work flow since it eliminates the need to continually switch modes to edit different parts of a mesh or scene. Moving the mouse highlights in orange the component that would be affected if you click making it easier to select the correct point or edge.

(P/E/F : Points/Edges/Faces)

Build 1143 dated December 10, 2014 implements a different mouse model. It affects the Point Editor's **Move**, **Rotate**, **Scale** and **Scale Non-Uniform** tools and works like this:

I. If Fast Select is enabled:

1. **Click without dragging (All tools):** Only affects P/E/F selection. Never changes geometry:

LMB: Select highlighted item. Deselect everything else. Clicking on empty space deselects everything,

MMB: Deselect highlighted item. Nothing else changes selection. Used to "Subtract From Selection".

RMB: Select highlighted item. Nothing else changes. Used to "Add To Selection".

2. **Click and drag:** *Affects selection and geometry.*

Selection

Selection (**Arrow Select**):

Same as click without dragging.

Selection (**All tools except Arrow Select**):

Clicking on a selected P/E/F doesn't alter selection. Everything currently selected is moved/scaled/etc.

Clicking on an unselected P/F/E selects only that item, everything else is deselected. Only that one item is moved/scaled/etc.

Clicking in empty space doesn't alter selection. All previously selected items are moved/scaled/etc.

Geometry

Geometry (**Move**):

LMB: Move in the X/Y direction (e.g. in the plain of the screen).

MMB: Same as LMB. Possible additional functionality here. **New in 1143:** moves points in the direction of the Normals.

RMB: Move in the Z direction (e.g. into and out of the screen).

Geometry (**Scale**):

LMB/MMB/RMB: Scales selected geometry around the geometric center of selected groups of Points/Edges/Faces.

Note: Isolated single points don't move. Think about it if this seems strange

Geometry (**Scale Non-Uniform**):

LMB: Scale connected groups in X/Y directions independently (screen aligned).

MMB: No effect.

RMB: Scale in the Z direction (e.g. into and out of the screen).

Geometry (**Rotate**):

New in 1143: Rotates around the geometric center of selected groups of Points/Edges/Faces.

LMB: Rotates in the X and Y screen axis.

MMB: Rotates in the direction of the average of the selected faces' normals.

RMB: Rotates in the Z axis (which points out of the screen).

Click-drag with <Ctrl> Key

Click-drag with <Ctrl> Key Pressed (All tools): New in 1143: Only affects P/E/F selection.

Never changes geometry:

LMB: Paint-select. Initially deselects everything, then selects as the mouse moves.

MMB: Paint-deselect. Deselects as the mouse moves over things.

RMB: Paint-select without delete. Same as LMB but doesn't do the initial deselection.

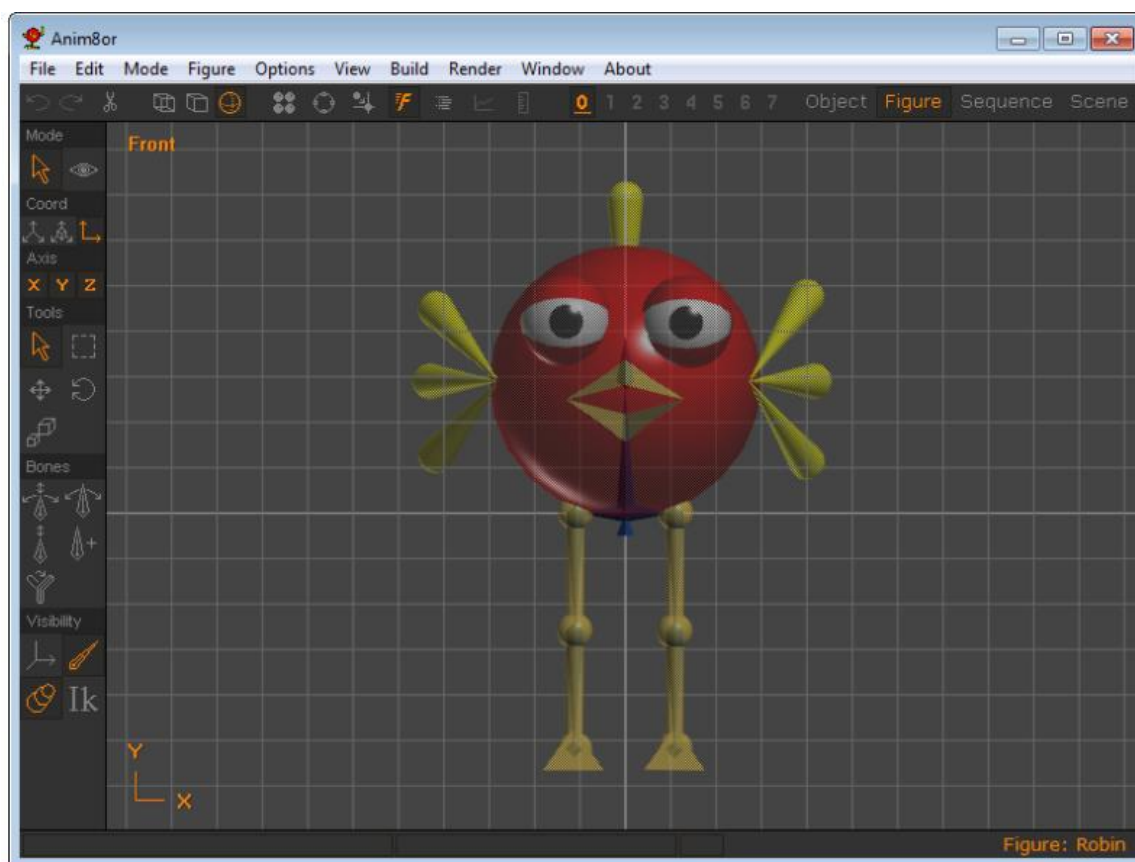
II. If Fast Select is disabled:

1. Keeps "Classic Anim8or" behavior.

Figure Editor

The **figure editor** is where you design animated characters to put into Anim8or's scenes. The main difference between a figure and an ordinary object is that a figure contains a **bone** structure, or skeleton, that you use to bend and shape it in different poses. By setting the pose of a character in a few **key frames** of an animation, you can bring it to life. Anim8or will smoothly fill in all the in-between poses by bending the joints of the skeleton just the right amount. You can enter the figure editor at any time by selecting **Mode→Figure** in the menu, or by clicking on **Figure** in the right part of the top toolbar.

The basic screen is similar to the Object Editor and is shown below. You can also view multiple viewpoints simultaneously, as with the other editors.



Notice the bones that are faintly visible inside of the character's body and legs. You can control the visibility of bones, your character's body parts, and other aspects to help with the design. This character doesn't have any particularly interesting bones in his body, but does have animatable legs and feet.

The mode section in the figure editor has two modes: **edit** and **viewpoint**. You click on one of the icons to switch between these two modes.



Figure/Edit [A] mode button. This is the initial mode that the figure editor starts in. You use it to do all of your character design.



Figure/Viewpoint[V] mode allows you to pan, scale, rotate, and size-to-fit any or all of your views of your workspace.

Figure Basics

There are two different parts that you make when building a figure: the **body**, which is all of the visible parts of your character that will appear in the final images and the **bones** or **skeleton** which you use to pose and animate your character.



You will usually design all of your character's body parts in the Object Editor as different objects. When you add them to your character you are only making a reference to the original object. This way you can add an object several times without increasing the size of your working files, and you can make changes to all the instances by editing a single object.

There are also a couple of "built in" shapes in the figure editor to help with preliminary design and motion tests. The "ball and stick" legs shown here are made from built-in cylinders and balls.



Once you have made your character's parts, you can switch to the Figure Editor and build a skeleton. Starting at the initial root bone, add bones one at a time to make the movable structure of your character. For example, if you want human-like legs, you need to add a bone for each of the thighs, lower legs, and feet. You can give each joint its own range of motion which makes animation a lot easier.

Once the skeleton is designed you can add the character's visible parts. Then link each part to a specific bone, and when that bone moves, so does the body part. You can also move, scale and rotate each part individually within the coordinate space of its bone.



Figure/Edit Operations

You use the first group of buttons to select various parts of your character, and to place the visible parts on the bones in a similar manner to the same buttons in the Object Editor.



Select [a] -You use this mode to **select** both your character's bones and body parts. As usual, you can use the right mouse to select additional elements while keeping the original ones selected.



Drag-Select [d] -This button sets **drag-select [d]** mode. You drag a rectangle on the screen and select all objects that fall within the final area. Again, using the right mouse button adds to the current selection.

The next three buttons apply to the objects that you have attached to the skeleton, not to the bones themselves.



Move [m] -You use this one to **move** an object around relative to the bone that it's attached to.



Rotate [r] -You can **rotate** objects with this button.



Scales [s] -This button **scales** objects. You have to select an object before you can modify it.

You use the next set of buttons to edit the bones of your character's skeleton once it has been built.



Edit Bone [E] -The **edit bone** tool lets you both rotate a bone and change its length. There are 3 distinct actions:

1. Click-dragging on the **rotation widget**'s controls rotates the bone according to what part of the widget you click but doesn't alter its length,
2. on the tip of **the** bone rotates the bone and alters its length to follow the mouse, and
3. on the bone itself rotates the bone around its X, Y or Z-axis with the Left, Right and Middle mouse buttons.



Rotate [R] -This mode allows you to **rotate** your character's joints into their **nominal** position. This position defines the zero value for all rotations that this joint can perform. It is not the same as the **default** position of the joint, which is where the joint is when it's "relaxed", but is used to define the bone's coordinate system.



Length [L] -You use this button to change the **length** of a bone. When it is pushed, you can click on a bone and stretch or shorten it with the mouse. You may click anywhere on a bone but you must first select it if **Fast Select [Ctrl-T]** is not enabled. You can also double click on a bone and set the length and other parameters in a dialog.



Add [N] -You use this mode to **add** new bones to a skeleton. To add a new connecting bone, click on the **end** of a bone and drag to set the bone's length. The new bone will align in the direction that you move the mouse.

You can also use the **Build→Insert-Bones** command to insert a new bone as the parent, and to specify a precise length.



Skin [S] -This button allows you to **skin** part or all of a skeleton with a single Object. (Skinning means to join the skeleton to an object so that they work together as one.)Then when the joints of a bone bend the object deforms and bends seamlessly along with the skeleton. There is more detail on how to use this skinning tool later in this chapter.

Visibility

You use the final group of buttons to control what is visible in the edit window. You may find it easier to manipulate your character's skeleton without its body getting in the way, or want to see the final, solid look without the bones showing. These controls will do just that.

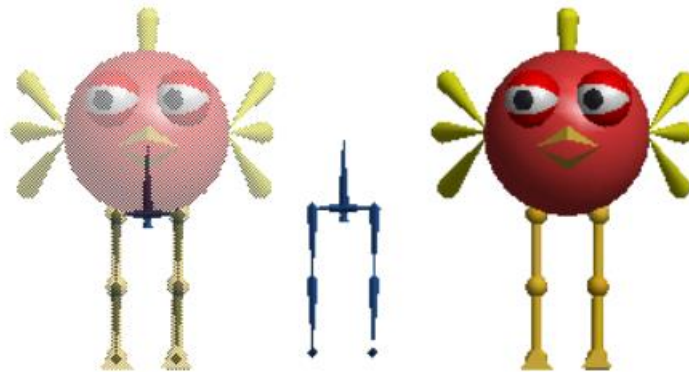


Bones [B]



Objects [O]

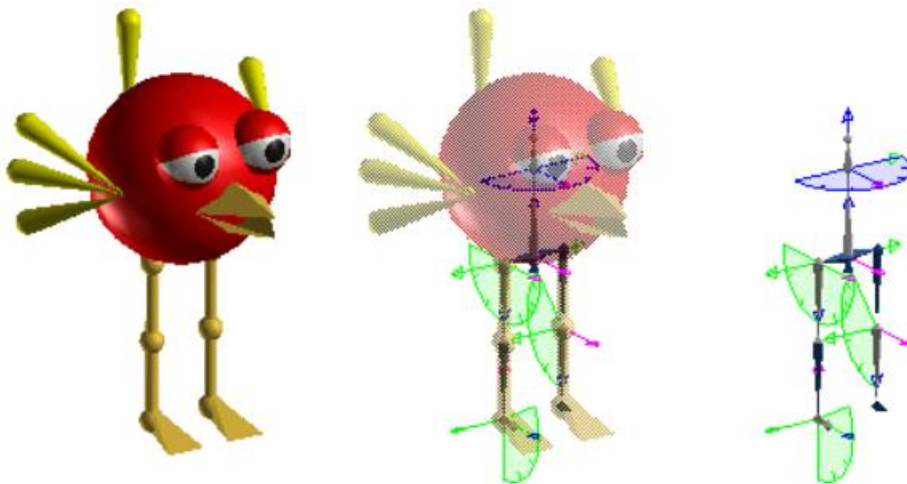
You use these two buttons to toggle the visibility of **bones** [*B*] and **objects** [*O*] or body parts. If both of them are enabled then your character's body will be shown in a sort of transparent view with the bones visible inside. You will find this indispensable when arranging your character's parts. If only one is enabled, the corresponding items are shown as solid objects.



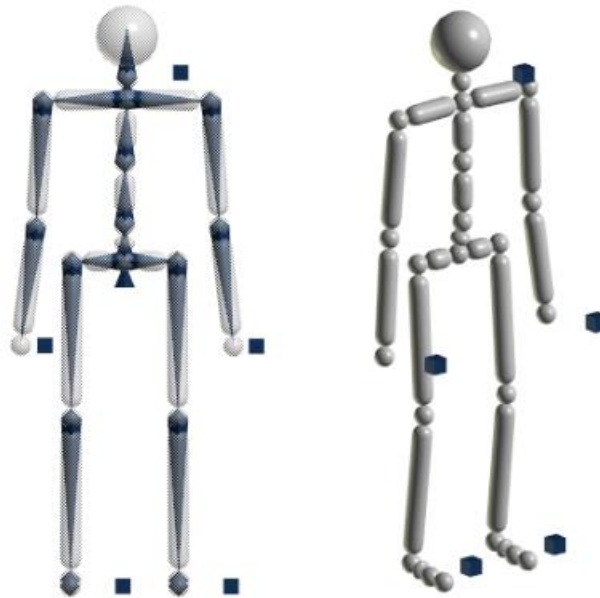
This shows a character with a simple skeleton. Only the legs have bones, so you could only animate the legs. The face would be frozen in that silly look unless you add additional bones to animate the eyes, wings, beak, etc.



Axis [*X*] -This button toggles a visible **axis** [*X*] for any selected bones. It also shows you the range of movement for their joints.



Inverse Kinematics [*I*] -The **inverse kinematics** [*I*] button displays your figure's IK control handles. You add IK chains by selecting the first and last bones in the chain and using the **Build→AddIKChain** command. Handles are shown as small blue cubes and are primarily used to select individual chains. You can move and scale them in the figure editor to make individual IK controls easier to select. Animation using IK is done in the Sequence and Scene editors.



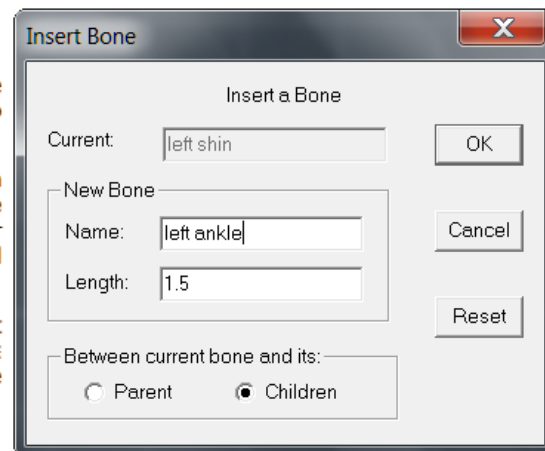
Building a Skeleton

Each figure's skeleton starts with a **root** ▲ bone. By adding **child** bones the root, and then to successive bones, you construct a skeleton one bone at a time. Each bone is owned by its **parent**, and any movement that the parent makes also moves all of its children.

When you add a bone it is aligned in the same direction as its parent. So you will normally have to rotate it to its proper orientation.

If you want to **delete** a bone, first select it and then choose **EDIT→CUT** from the menu bar. This will delete the bone and all objects attached to it. Any lower bones will be moved up and attached to the deleted bone's parent.

You can also **insert** a bone into your figure. First select a bone and then use the **BUILD→INSERT-BONE** command. You can insert the new bone either as the parent or as the child of the selected bone.

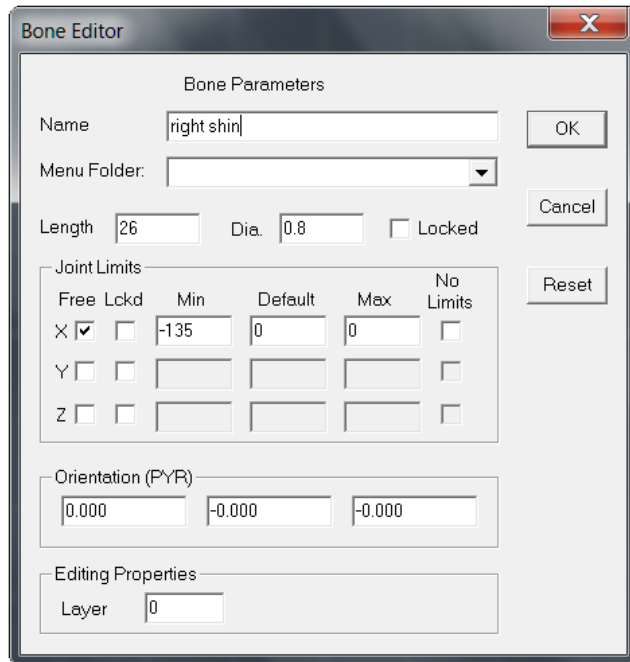


This is a simple skeleton. It is made out of 9 bones plus the root. This skeleton is built with two symmetrical hips and legs. The lower two bones in the right leg are selected and shown in white.

The foot has an extra diamond ♦ visible on the end. This gives you an easier target to grab onto when you are manipulating the bone. The lower leg has one as well, but it is hidden inside the foot bone and isn't visible.

Flexible Joints

In Anim8or you decide which joints are **flexible** and which ones are frozen in a **fixed** position. By default they are all fixed. You add flexibility in the **Bone Editor** dialog that appears when you double click on a bone.



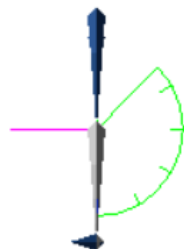
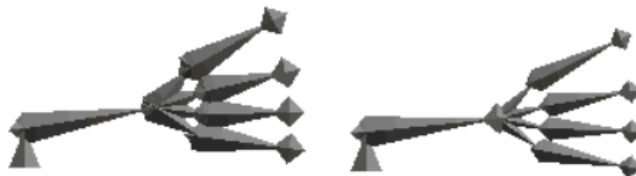
You control the rotation of a joint around each axis independently. Clicking on the "free" check box for an axis allows it to rotate around that axis. You can set the minimum, default, and maximum allowable angles as well.


You use the **X** and **Z**-axes to control the bending of a joint, and the **Y**-axis to twist around. Clicking the single "Locked" check box will prevent any changes to a bone's length or position in the basic skeleton. It's a good idea to lock your bones once you've got things set up. This prevents their accidental alteration. You can still change all the joint parameters, but since this is done in a dialog instead of with the mouse, it is less likely that you will do something accidentally. The "Lckd" check box for each axis is used when you are animating a character and doesn't apply to the figure editor.

Angle values are limited to values between **-359** and **359** degrees with a maximum difference of 359 degrees.

If you check the "No Limits" box then you will be able to rotate the bone freely in that axis. This kind of joint is called a free or unconstrained joint because there are no restrictions on its position.

Working with a lot of small bones, such as in a hand, can create a confusing jumble of overlapping bones. You can change the diameter used to show each bone in the Bone Editor by changing the value in the **Dia.** field to help clear up the view.



This is a side view of a skeleton. The show axis  [X] button is set so you can see the joint rotation limits of any selected bones. The x-axis is shown in green the y-axis in blue and the z-axis in violet.

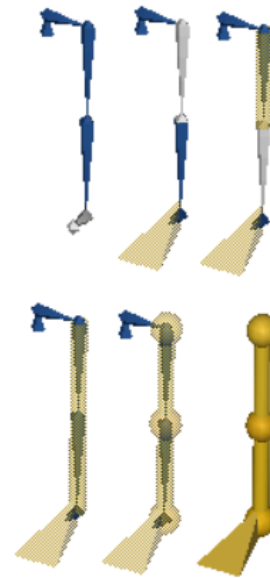
The bone selected here is the lower part of the right leg and forms the right knee. It can only bend in the x-axis and like a human knee can only bend backwards in this case a maximum of 135 degrees backwards.

Adding Body Parts

Once you have built a skeleton, you need to add the visible body parts that will appear in your final images. Each body part is an object that you add to a specific bone's local coordinate system. First select the bone that you want to add your object to, then use the **BUILD→ADD-OBJECT** menu command to add it. Your object then behaves like that bone. When the bone rotates, so does the object.

You can also add simple cylinders and spheres. They aren't very sophisticated, they just look like sticks and balls, but they do allow you to do quick tests to see how your skeleton behaves. You add them using the **BUILD→ADD-CYLINDER** and **BUILD→ADD-SPHERE** menus.

You can change a few of the properties for these simple shapes as well. Double click on one to bring up its property dialog, and use the **SETTINGS→COMPONENT→MATERIAL** command to set their color. These basic shapes are not as flexible as Objects that you build yourself. They have some fairly restrictive limits and aren't meant for any kind of final rendering. For finer control than this you need to build things in the Object Editor.



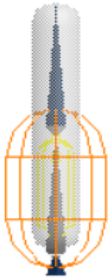
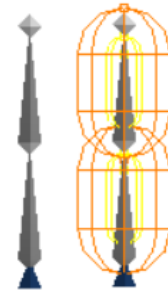
Skinning


In the previous section you've learned how to add an Object as the offspring of a specific bone. It follows all of its parent bone's movements, but remains a rigid, inflexible Object. This is what you want when making a robot. It's not exactly what a living human or animal does. You want them to be flexible, to change shape and bend along with their skeleton. This is exactly what **skinning** does. Once you've built an Object and a skeleton, you can attach your object to several bones at the same time with the Skinning tool. Here you set how strongly each bone controls each point on your Object. The stronger the influence the closer it will follow that bone's movement.

There are two ways you can set the level of influence. The first is to use the **influence volume** that surrounds each bone. Any point in a skinned Object that lies within this volume is bound to that bone to a certain degree depending on how close it is to the bone. You can use this influence to animate your object so that it bends along with a Figure's skeleton. Alternately you can paint your own **skinning weights** to bind your Object to a Figure.

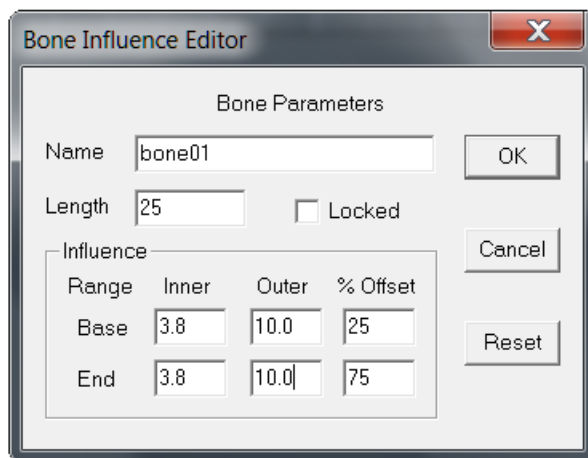
Influence Volumes

The two bones at the right show their influence volumes or ranges. The inner yellow area is where the bone exerts maximum control. This does not mean that other bones can't also control points that fall within this region, just that the bone is a strong influence. The outer orange area is the limit of the bone's control. That bone does not affect points that lie beyond this boundary at all. The only exception is that if a point lies outside all bones' areas of influence, it is simply controlled by its base bone, the one that it is initially attached to.



To skin an object you first attach it to a bone called its **base** bone. This is the default bone controlling all points in the object but all bones in your figure are capable of influencing the shape of your object. Then enter Skinning mode by pressing the skinning button on the toolbar  [S]. Now select your object by clicking on it. This will enable skinning for your object and add its base bone to its influence set those bones which can alter its shape. To add other bones simply click on them and confirm that you want to add them in the dialog that appears.

Bone Influence Editor

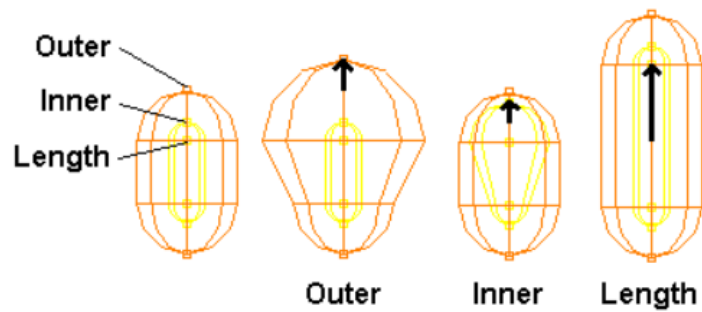


You can adjust the size of a bone's influence in two ways. You can double click on it and enter numeric values, or you can edit the influence regions directly on the screen.

You can set the size of the inner and outer range for each end of the bone. You can also set the position along the bone where they start and end as a percentage of the bone's length.

Changing sizes directly on the screen is often more effective but it can sometimes be tricky to do because things are often cluttered with several overlapping regions.

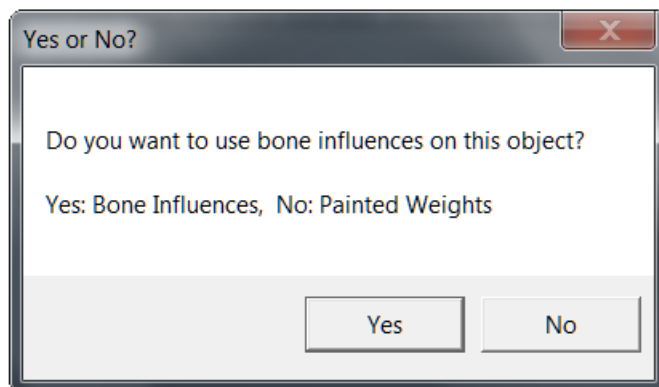
There are six control points on each bone, three at each end. If you turn off the visibility of both bones and objects by clicking on the buttons at the bottom of the toolbar you will get a clearer picture of what's there. You use the one at the tip of the orange area adjust the size of the outer region, and the one at the tip of the yellow area for the inner one. The inner yellow point adjusts the offset.



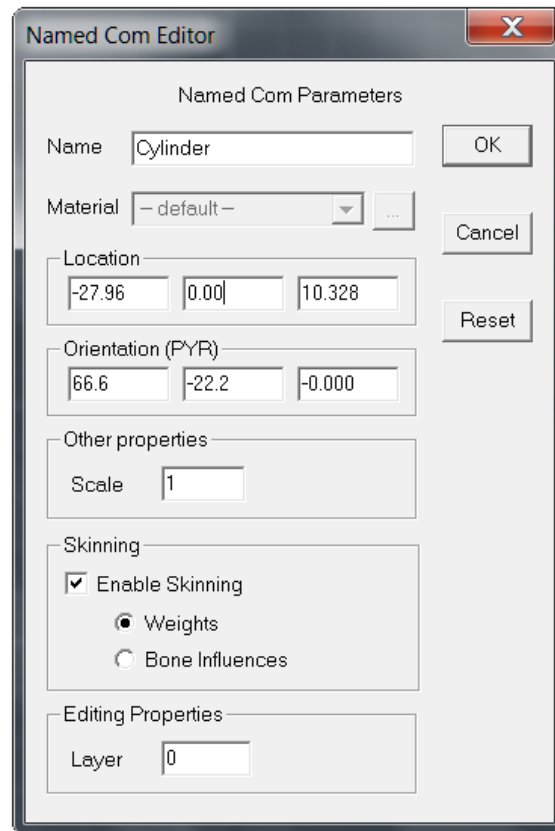
One final note: Don't attempt to adjust your bone's positions in the Figure Editor and expect to see your mesh bend! It doesn't work that way. You will only be adjusting the bone's position, and area of influence, relative to your object. You must go to the Sequence or Scene editor to start animating your character.


Skinning Weights

For more control you can also paint skinning weights directly on your objects. It's best to start with a basic bone influence volume weighting and then convert to using weights.



You can change to weights by double clicking on an object and selecting the Weights button in the Skinning area of the parameter dialog.

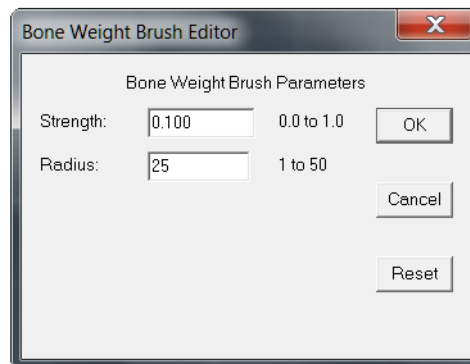


To paint weights enable skinning  and select an object. The display will show the object's bones in contrasting colors. You then right-click on a bone to paint its weight on the object.



The brush is a green circle with a cross in the center. Hold the left mouse button down and drag the brush across your object and the influence of the bone will be increased for the points that you paint. The color of the object will change to show the new weights.

The center of the brush has the most "paint" and will make the biggest change in weights where it passes. It decreases to the edge where the change stops. You can change to size and strength of the brush with the **Build→Weight-Brush** command.

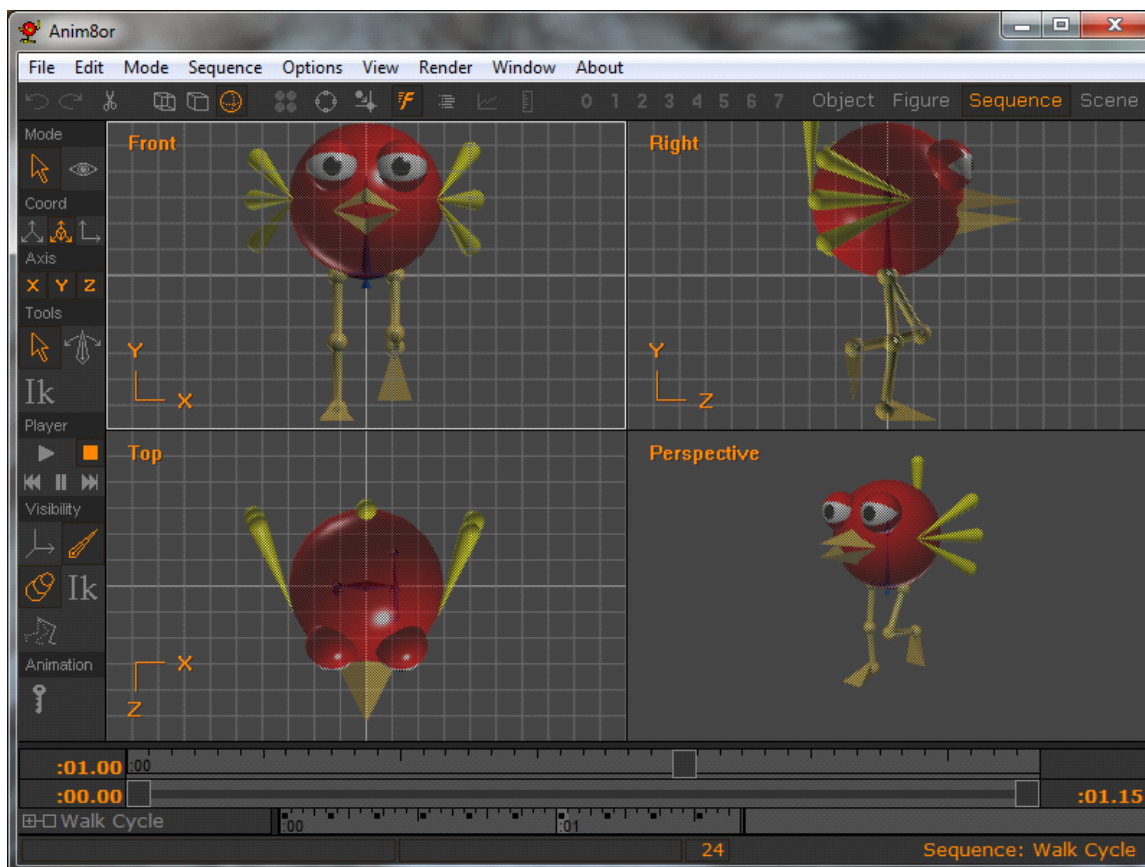


Now you're ready to go to the Sequence Editor to make some animated sequences. Then on to the Scene Editor for your final movie!

Sequence Editor

The **Sequence Editor** is where you can create movement that gives personality and style to your characters. A **sequence** is a short block of key frame poses, such as one cycle of a walk. The easiest way you can animate a figure is by placing it in a scene and adding one or more sequences.


Longer animations can be built from several short sequences, such as building a long walk out of multiple applications of a walk cycle. Anim8or automatically joins the adjacent sequences into a single, smooth movement, using the key frame positions as guides. Here is a screen shot of the Sequence Editor in action:




Time Track

The **time track** at the base of the Sequence Editor indicates the current frame shown in the viewports. The frames that have one or more joints set in key positions are marked with small black squares ■ in them. The numbers **:01** show the time in seconds since the start of the Sequence of a position on the track. The current frame is highlighted with a lighter gray background.

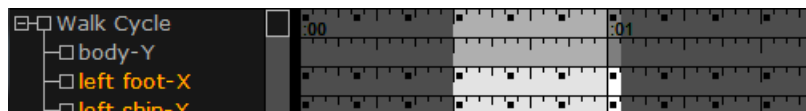


You can expand the track to see a tree view showing details of individual joint's keys by clicking on the plus symbol  next to the title.



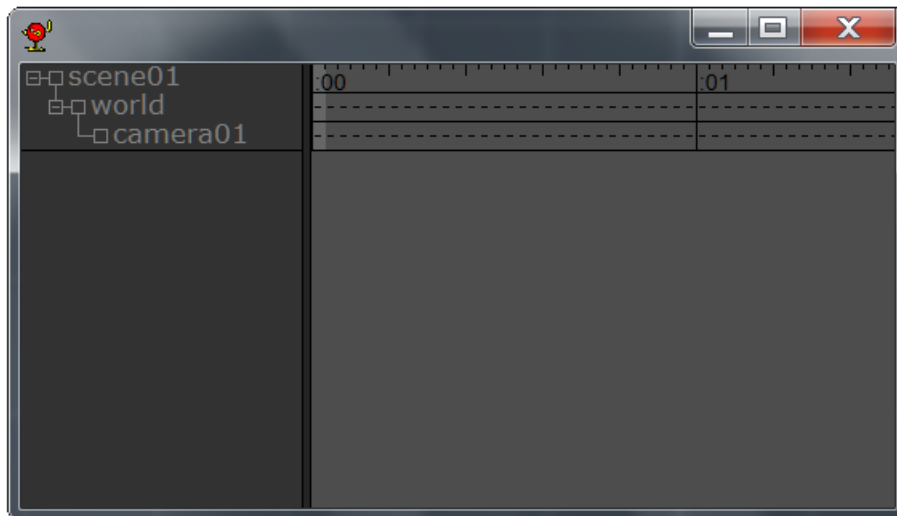
Key positions are shown for each joint/angle with a small black square in the frame. You can also select a joint by clicking on it in the expanded track. Resize the expanded track view by clicking and dragging the vertical resize bar  at the bottom of the toolbar up or down.

The track will automatically scroll to keep the current frame visible when a sequence is too long to show it all at once. If you want to see a frame that is off the window, you can either use the **Sequence**→**Properties** (earlier versions: **Settings**→**Sequence**) command to set the frame numerically, or you can use the Left or Right Arrow keys to play the sequence forwards or backwards until you reach your frame.



You can move to a particular frame in a sequence by clicking on that frame in the track. If you click and drag the mouse it will select a range of frames. Clicking on the name of a bone selects that bone and highlights it in the track window. Clicking on the title of the sequence selects all bones. Mouse buttons have their usual meaning on the bone names: the right button adds to the selection, and the middle button subtracts. Use Shift-Click to select a new frame range while still keeping the same current frame.

If you prefer, the Time Track now has the option of becoming a floating window in both the Sequence and Scene modes. You can find this under **Options**→**Float Time Track**.



Floating Time Track

Scrubber Bar

It's often useful to play your animations freely forwards and backwards to check motion and for tweaking animation. This action called **scrubbing**. Anim8or's sequence and scene editors have a tool call the **scrubbing bar** that allows you to do this, unlike back in the good old days when you

had to flip sheets of paper to do this :).

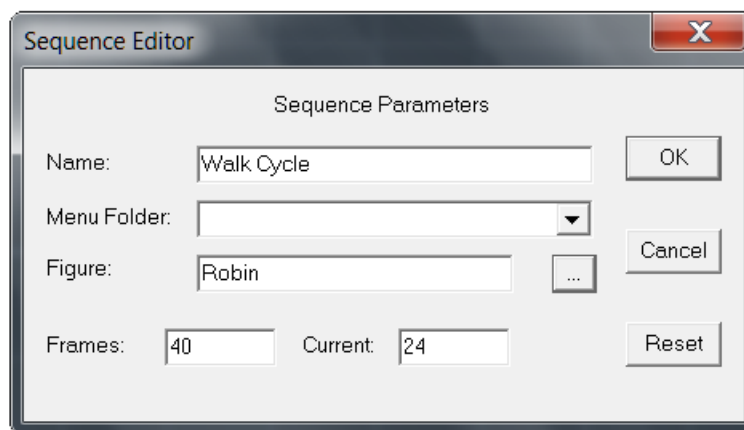
You can click on the button in it's **top** row and slowly or rapidly drag the mouse right and left and the animation will run forwards and backwards in response.




The **bottom** row allows you to set the limits of the scrubbing range. You can click-drag the two end buttons to adjust the range.

Sequence Basics

You can't animate a character before you have one! So it isn't surprising that you must first design a character in the Figure Editor before you can give it any movement. Once you have one, you switch into the **Sequence Editor** and a new, blank sequence is created. Select the **Sequence**→ **Properties** command to show the Sequence's settings dialog:



Click on the  button and select your figure from the list. You can also set the number of frames in the sequence in this dialog. Now your figure will appear in the viewports and you can begin to animate it. You can also set the Sequence's length and set the current view to a particular frame with this dialog.

The sequence editor has two modes: **edit** and **viewpoint**. You click on one of the icons to switch between these two modes.



Sequence/Edit [A] mode is the initial mode that the sequence editor starts in. You use it to set all of your poses and to view the motion of you've created.



Sequence/Viewpoint [V] mode allows you to do the usual pan, scale, rotate, and size-to-fit any or all of your views of your workspace.

Edit Operations



[A] You use this button to **select** bones in your characters. The right mouse button allows you to select multiple bones



[R] This mode allows you to adjust the **angle** of your character's joints to new positions and for new key frames.



[I] In the Tools section of the toolbar, this is the button for **inverse kinematics**, a helpful tool for posing and animating your characters.

Visibility Options

These controls allow you to show different aspects of your character on the screen when you are working.



[X] This button toggles the visibility of the **axis** and the range of motion of any selected joints.



[B] This button controls the visibility of the **bones** of your character, like in the Figure Editor.



[O] This button controls the visibility of the **body parts** of your character, like in the Figure Editor.



[i] In the Visibility section this button toggles the visibility of the IK handles when you are not actively editing them.



[G] Use this **ghost view** button to show multiple frames of a sequence simultaneously. You can compare previous key poses and in-betweens that Anim8or creates with another frame's to help see how they fit.

Animate Button

These controls allow you to show different aspects of your character on the screen when you are working.



[K] This is the **animate** button. It is your magic wand for making things move. You use it to add and change key positions. When you change the angle of a bone you sometimes want to do it for all frames in a sequence, and sometimes just in the current frame. The key button gives you this choice. If it is not pressed changes apply to the entire sequence. If it is pressed only the current frame is changed. The next sections explain this in more detail.



What is a Key?



If you had to set the position of *each* joint of your figures in *each* frame of an animation, that's about all you'd ever do! Fortunately there is an easier way to animate called **key frames**. With it you pose your character in a few frames that represent what it should be doing in that particular frame, and the computer smoothly fills in the in-between frames. You get the best results, naturally, if the frames you choose represent the significant points in your character's movement. That's why they're called "**key**".

There is no reason that all the joints have to be a key in the same frame. If your character is walking forward and slowly turning his head you may want dozens of key positions in the legs but only a couple in the head turn.

Making a Key Pose

When you first create a Sequence, your character is shown in its default position. You can

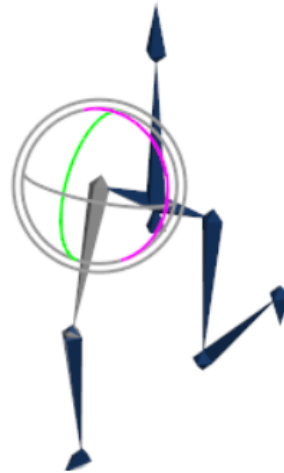
change its position by bending its joints. But you can only bend the ones that you made flexible in the Figure Editor, and only in the ways that you said they could bend.

To make a key pose enter Rotate Bone  [R] mode, then enable Animation  [K] . Select the joint or joints that you want to move, and then you can either use the trackball or rotate the bone directly to pose your character.

Using the Trackball

The most recent selected bone will display a trackball centered on its starting joint. The axes that are free to rotate are shown in color on the trackball. Click-drag on the axis that you want to rotate to add a key position for that joint.

If an axis isn't highlighted in color it can't be rotated either because it isn't enabled or it's too difficult to edit from that angle.



Trackball Rotation for Bones

Steve

Administrator

Hero Member

May 02, 2015, 01:07:20 am

The Bone Rotate tool in Build 1175 uses a trackball. You can rotate bones in 6 different ways with just the left mouse button (**NOTE:** in Anim8or 1.01.1318 any mouse button can now be used.).

The colored arcs rotate the joint around an axis in the bone's local coordinate system: Green for X, Blue for Y and Violet for Z. Click-drag in the direction of the arc to rotate the bone.

The outer ring rotates the joint around the screen's Z axis, and the Yellow arrows rotate it around the screen's X and Y axes.

Note: Priority is given to the trackball controls, so clicking on a bone that's mostly covered by these controls can be a bit tricky to do. That shouldn't be a big problem because the controls should be as easy or easier to use. But allowing this makes it possible to click-drag on an unselected bone and rotate it without first having to select it, so I thought that this would be useful to support.

Give a big Thanks! to **raxx** for his feedback on the design.

You can also click directly on any bone and rotate it with the mouse just like the previous tool worked.

Click-dragging on Bones


Click-drag on a bone to rotate it directly. Anim8or will hide the trackball and display a transparent arc that shows the limits of rotation possible.

Each mouse button rotates around a different axis.

To rotate around the X-axis, which is shown in **green**, use the left mouse button. The joint will try to follow the mouse.


To rotate around the Y axis shown in **blue** use the right button. Since the Y-axis rolls the joint around the bone's length, and doesn't change the direction it's pointing, you have to move the mouse to the left or right to move the joint.


To rotate around the Z-axis shown in **violet**, use the middle mouse button.

Click on the tip  of the bone to rotate it in a more general way.

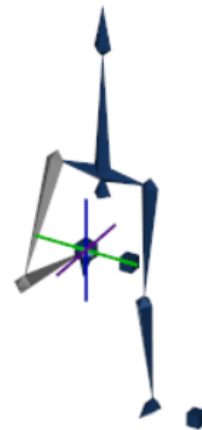


Inverse Kinematics

If you added any IK chains to your figure in the Figure Editor, you can key all of its bones in a single action. Enable IK mode  [**Ik**] then click on the IK chain's handle, a small blue cube, to display the IK widget and the End Effector. You can now move the end to the chain along one of the figure's axes using the colored lines, or in the plane of the screen with the end effector.

That's all there is to it! You've created a key position for a joint. To make your character move, go to another frame (click on it in the track window) and make a second key position. Click on the play button  [**Cr**] to see it move.

Continue setting the positions of any joints that need changing until you are satisfied with your pose.



ForumNotes:

Updated IK tool in build 1247 dated September 4, 2016. The controls have been improved significantly. Here's the updated details:

To add an OK chain: In the Figure Editor, select two bones, one which is the ancestor of the other. Then apply the Build->AddIKChain command.

IK Handle: The IK Handle is shown as a small cube when the IK button is enabled in the Visibility section of the left-hand toolbars in the Figure editor. You use this to access the IK chain's properties here. Double click to **edit**, **move** and **scale** it with the normal tools.

IK Widget: In the Sequence editor you move IK chains in IK mode. In addition to the IK Handle, the End Effector and move widget are shown. The **End Effector** is a small icosahedron positioned at the tip of the tail bone of the IK chain. Centered in the End Effector is a 3-axis cross hair for posing the chain.

IK Goal: The goal is the location that you are trying to position the end effector. As you move it

around the joints in the associated IK chain will bend to try and keep the end effector aligned. If you move the goal out of range Anim8or will get as close as it can.

Undo: should work for all IK related actions. Please report anything that doesn't work correctly.

To use IK:

1. Add IK controls to a Figure by selecting two bones, one which must be the parent of the other, and use the Build->AddIKChain command.
2. Create a Sequence with the Figure and click on the IK tool. The IK widgets will be shown and the animation key will be automatically enabled.
3. Click-drag on the End Effector to move the Goal in the plane of the screen. You can also click-drag the IK Handle for this. Click-drag on the different axes to move the Goal in the X, Y or Z direction relative to the Figure.
4. If the goal is unreachable, it will be shown separately from the end effector with a dashed line connecting them.
5. Anim8or adds keys for all joints in the IK chain when you move the end effector. All joints are then animated according to their individual keys. This is not the best behavior. I think a better way is to animate the end effector and use IK to position the joints accordingly at each frame. But what I have ready now is a good start

New to build 1247

Locked End effector Position: In the Scene editor you can lock the position of an end effector. Then when you move any part of the figure between the locked joint and the root the position of the figure will automatically adjust to keep the effector in the same location. **To lock an effector:** double click on the handle and check the Lock Position check-box. *Note: currently only one effector can be locked at a time.*

Full Body IK: The entire body of any figure can be manipulated. Click-dragging on any bone or attached object will create a temporary IK chain from that bone to the root. **This is an experimental feature.** There are some issues, such as the temporary effector only moves in the plane of the screen so if you aren't in the proper view it will distort the figure. Of course the normal IK tools still work the same. Please try this out and let me know what you think.

Selected IK Controls: Build 1247 doesn't show the IK widget until you click on the handle to select it. This reduces the clutter on the screen with multiple IK chains. It also makes easier to see what is happening because the IK widgets are now drawn on top of everything. However when several IK handles are selected the widgets can appear incorrectly drawn.

Future details: restraints on orientation of joints, IK chains that don't have to be ancestors, different IK solvers.

Joint Limits: using limits can help when you are animation. IK favors the default angle and obeys all joint limits. Without limits you can get some strange results.

Key Selected Bones (hot key 'k') . When you click on an IK handle it automatically selects all bones, but doesn't add keys unless you move the mouse. Then you can use the 'k' hot key to key them as they are. I also added Morph Targets so "Key Selected Bones" is now "Key Selected Bones And Morph Targets". I also add entries in the track window for all joints and morph targets even if they don't have any keys set.

Key All Bones will key all bones in any figure that is selected or has at least one bone selected. Note: switching between modes will preserve figure and bone selection, but entering IK or FK

will deselect everything else (cameras, attached objects, lights, etc.) since these other things aren't relevant for these operations.

(December 20, 2016)

An update on the jitter: I've created some examples that are extremely sensitive to jitter. I'm now experimenting with various damping schemes and alternate IK algorithms.

(December 28, 2016)

Check out build 1264 for a new IK solver! I implemented several different solvers. To see how they behave, try the different scenes with the attached project. The three solvers used are:

1. **Jacobian Inverse**. Supposedly the best. Very fast solver but becomes quite unstable when a target is unreachable. Two jointed chains do well but 3 joints shows problems. See scene "Tripple Chains" (yes, it should be "Triple" but I don't want to rename everything just to fix my bad spelling). The first 15 frames are stable but after that the effector moves to the side which the figure can't do. This causes jitter.

2. **Jacobian Transpose**. This is the best. It's can be slow to solve, especially when the effector is out of reach. Anim8or uses it now.

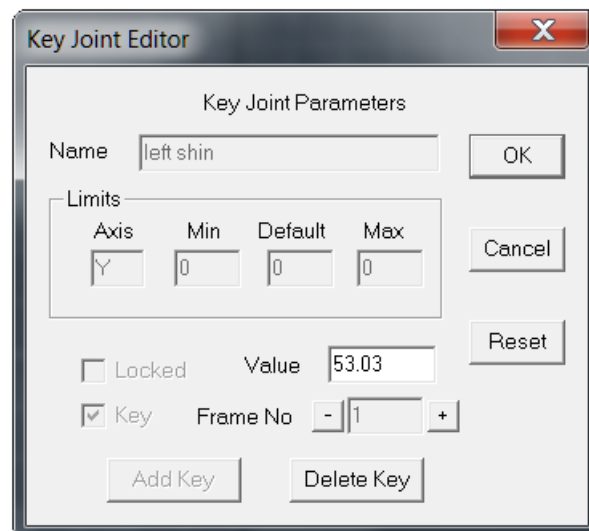
3. **Monte Carlo**. An alternate solver works OK with 2-bone chains with limited joints. Also work when the target is out of reach. Otherwise it's very jumpy. Left it in for this build just for fun.

Note: The only solver supported at the user level is **Jacobian Transpose**. You won't be able to use the others in normal use.

Have fun!

Editing Key Frames

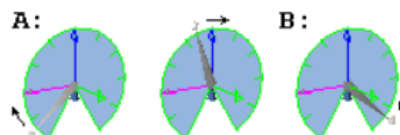
You can edit key frame values for individual joints. Double click on the joint's track in the expanded view of the track and the key frame editor dialog will appear:



The current value of this joint angle is shown in the *value* field, and the current frame next to *Frame No.* If this is a key frame for this joint you will be able to enter a new value for the key, and you can delete the key with the *Delete Key* button. If it isn't, you can make it a key with the *Add Key* button. The plus + and minus - buttons advance the current frame by one in either direction.

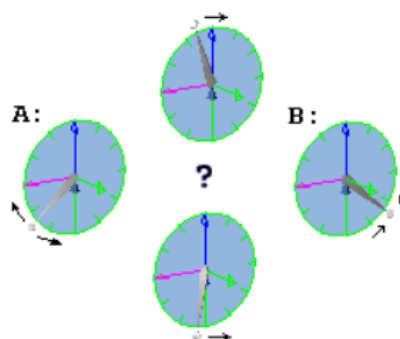
A Circle Has 720 Degrees

Sometimes rotation is not quite as simple as giving the first and last positions and letting the computer figure out what to do in between. For **constrained** joints (those with limits on how they can be rotated) like the one at the right, things are simple. Moving from position A to position B simply rotates to the right.




However for a **free** or **unconstrained** rotation this doesn't quite work. There are two directions that the joint can move: clockwise or counter clockwise.

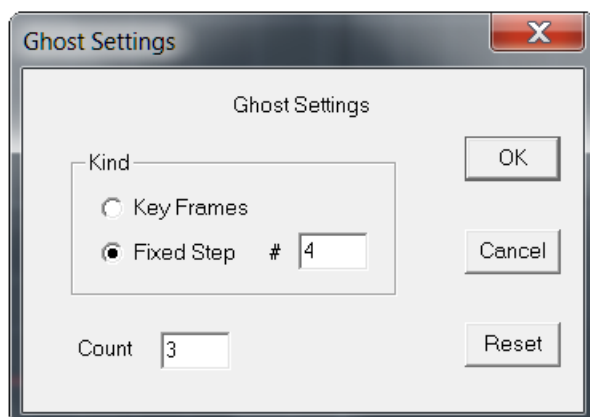
Anim8or resolves this dilemma by ex-panding the range of angles that you can use to be from -360 degrees to +360 degrees. There are then two values that represent the same position. One-quarter turn to the right is either +90 or -270. If the difference between the angles is less than 360 degrees, the joint simply moves between the two. If it's greater than 360 degrees then the joint moves the opposite way. It's not really as confusing as it seems once you've tried it out a time or two.



Ghost Views



If you enable Ghost Views  [G] you can see the relative positions of other frames. Then it's easier to adjust things relative to one another. This view can sometimes get rather confusing. It can help to hide body parts and just animate the skeleton.



You can change which frames are ghosted with the **OPTIONS→GHOST- SETTINGS** command. They can be limited to just key frames, or to those a fixed number of frames apart. You can also adjust the number of ghosts shown.

Making a Sequence

A Sequence can be thought of as a bunch of Key Poses played back in order. Each joint's position from one key frame is linked to the prior and following positions. Anim8or fills in joint positions for frames in-between key poses. When viewed this way, your character appears to the eye as if it is making one continuous motion. The smoothness and speed of the motion depends on how fast and in what ways the poses change. A large part of the work in animation concerns getting the motion just right.

After you finish the first key pose, set the sequence to a later frame by clicking on the appropriate spot in the track, or by using the left and right arrow keys. Then make a second key pose. You don't have to set the positions of all the joints, just the ones that need it. If one part of your character is moving at a slow pace, or needs less detailed control, save yourself some work. You can always go back and add more keys if you find you need them.

Exporting and Importing Sequences

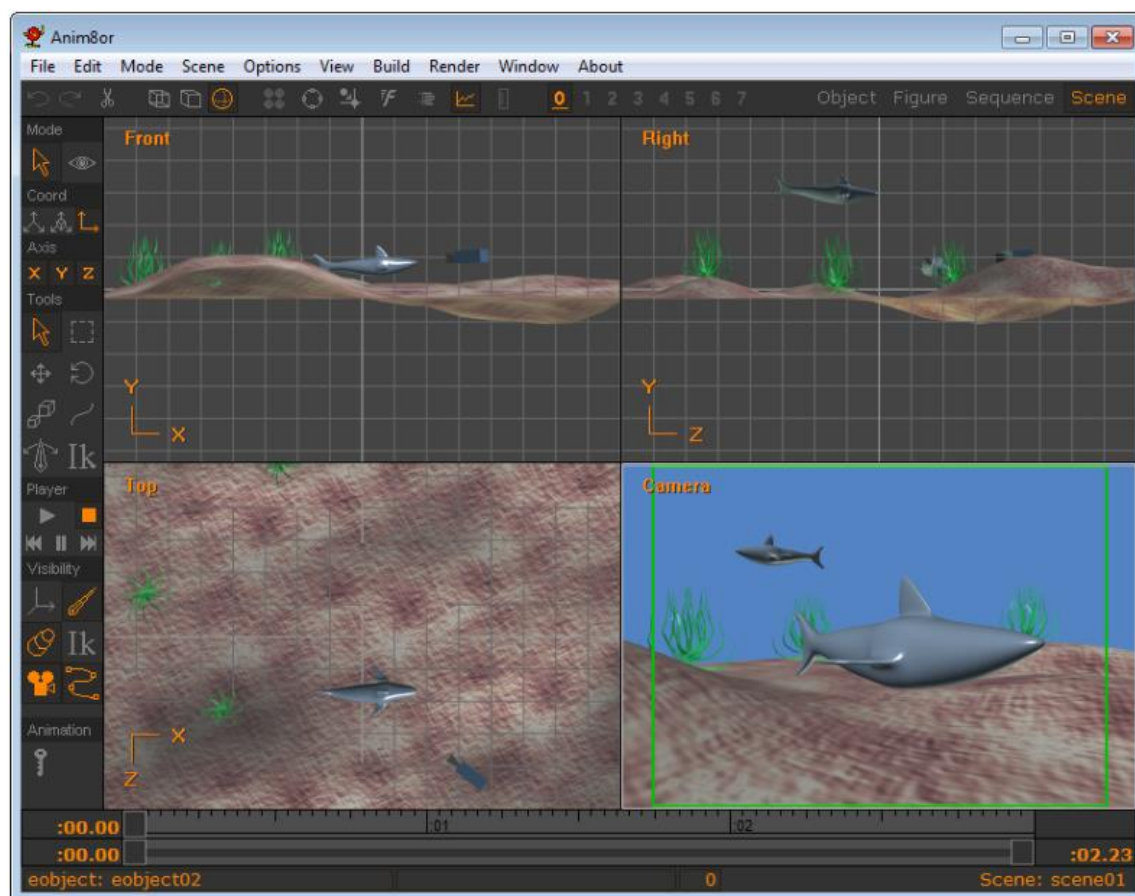
When you are satisfied with your sequence make sure to save it with a unique name. You can then import it into your scene to animate your figure. It will also enable you to modify your sequence in many ways (saving after each adjustment using a new unique name) so you have several related sequences that can be used later. A good example of this is a hand sequence that will allow you to have multiple differing hand sequences for your final animation and saving you much work. Saving them will also enable you to use them in another project using the same figure.

You can export them by going to **Sequence**→**Export** where you can save it as either an 3D Studio (*.3ds) or Anim8or (*.an8) file. You can import them by going to **Sequence**→**Import** where you can import it as either an Anim8or (*.an8) or Any (*.*) file.

Scene Editor

(This chapter, Chapter 7: Scene Editor is about the *editor*. For *animating* in the Scene Editor see Chapter 8: Scene Animation)

The **Scene Editor** is where you build sets, adding objects and figures, cameras and lights, and construct your final scenes. Everything can be animated; cameras, lights, and characters. And you can save the output as a sequence of still images, or as an **.avi** movie file. You can even generate stereo views. You enter the Scene Editor from the **Mode→Scene** command, or by clicking on the Scene tab to the right of the top toolbar..



The basic scene editor is shown above. You can configure it to show 4 views (front, top, side, and perspective), a single view from either of these views, an orthographic view, the camera's view, or combinations of 2 or 3 views. The usual menu, toolbar, and status bar are there, and a track bar that shows the current frame.

In addition to the normal select, move, rotate, and scale buttons in the toolbar, there are several more that perform functions specific to the scene editor.



VCR Block -You use the **VCR** style block of buttons to control the playback of your scenes. They work in the usual fashion.



Axis [X] -The **axis** button makes a joint's rotational limits and axis visible when you are animating joint positions.



Show Bones [B] -The **show bones** button controls the visibility of the bones in your figures.



Figure Objects [O] -You can show or hide your figures' bodies with the **figure objects** button. This can make editing joints much easier



Inverse Kinematics [i] -The **inverse kinematics** enables editing of IK chains.



Camera [C] -The **camera** button controls the visibility of your camera and lights. They aren't shown in the camera's view, the one that you normally use to render scenes.



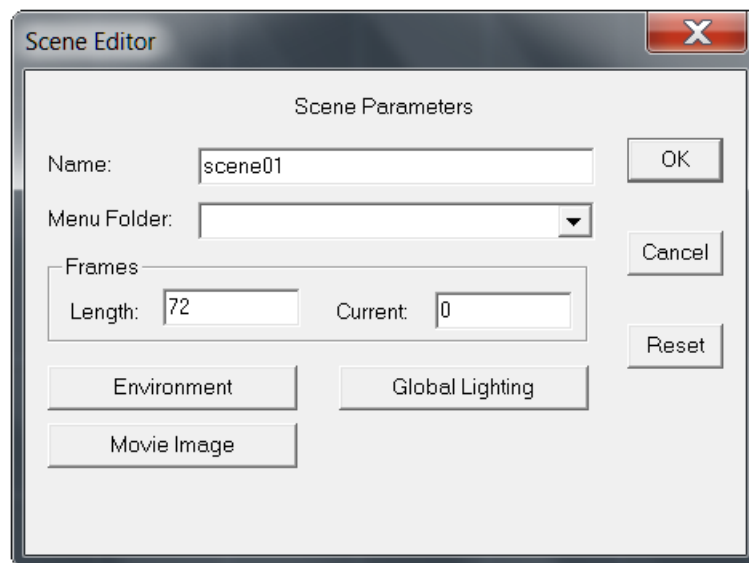
Path [P] -The **path** button sets the visibility of object's paths. Any object that you select will display its path as an editable spline.



Animate [K] -The **key** or **animate** button enables animation. When it's pressed, the changes you make to the location, orientation, size and other aspects of objects in your scene will be animated when you play the scene.

Scene Parameters

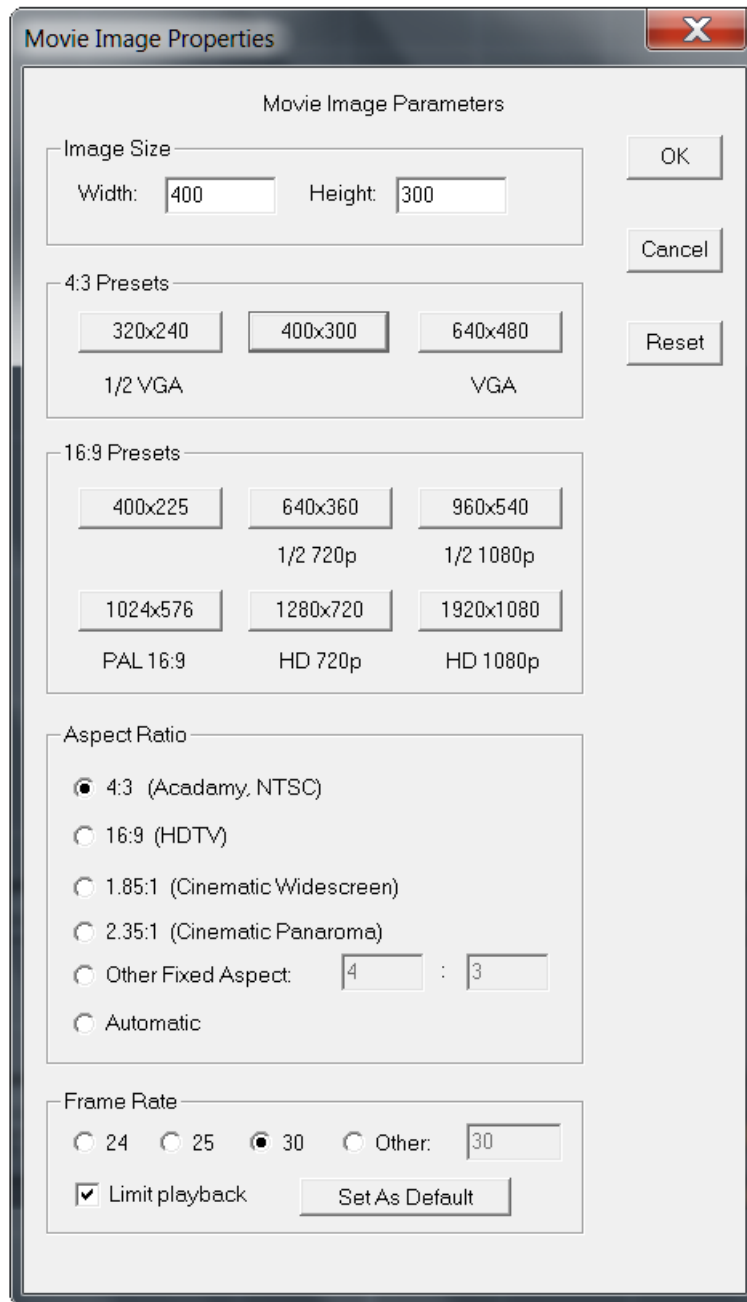
While they can be set or changed at any time when you are setting up your scene it is a good idea to decide on the basic scene parameters when you first begin. These can be found by going to **Scene→ Properties**.



Scene Parameters Editor

Here is a description of what you find in the Scene Parameters Editor.

- **Name** - Give your scene a unique name but keep in mind on whether this will be a stand-alone image, a single movie, or you will be using a separate movie editor to combine your movie shots together. In the latter case it would be best to give it a descriptive name to avoid later confusion.
- **Menu Folder** - You can name a virtual folder for the elements of your scene to be placed in, however make sure to save or export your scene to keep it.
- **Frames - Length:** Here is where you can set the length in frames of your movie. If you are only planning on rendering an image this input is irrelevant and can be ignored. However for an animation this number can be estimated once you have determined the length of your shot (in seconds) and the frame rate (in frames-per-second) you will be using. [length of your shot x frame rate = # of frames]. Currently the default frame rate is 30 but depending on your final use you may decide to change this. [*Hint:* except for long pans or fly-thrus the average length of a feature shot is on average around 11 seconds. Use a clock with a secondhand to give you a rough idea of the length you may need. This can later be adjusted if you find it a bit off.] **Current:** Shows you the current frame that you are on . This allows you to go to any frame by placing the appropriate frame number in and pressing OK.
- **Environment** - These conditions are set in the **Environment Editor** which is described later in this chapter.
- **Global Lighting** - This is discussed in **Chapter 11: Art Ray Tracer**.
- **Movie Image** - The **Movie Image Properties Parameters** dialogue box is where you set your movie settings. These settings are transferred over to the renderer when it is the actual time for rendering. Also, here is where you can set your frame rate. Please note that when you play back your movie it will race through quite quickly so make sure to set the **Limit playback**. This will slow it down to the actual time parameters you wanted. It is set to on by default. The **400 x 300** and **400 x 225** settings listed are for your quick previews to test how your movie will look and give you a chance to change something before committing to a longer render time.



For experienced filmmakers, videographers, animators, and those in the broadcast industry the terms, abbreviations and numbers shown in the parameters box are quite well known. However for beginners it may seem a bit daunting to try to figure out what it all means and what settings to use. When you are starting out don't be afraid to use any or all of them as you want. In fact, you may even want to try out your own settings just to experiment to be used for other purposes. However, if you decide to try to create a movie for a specific purpose such as a project for sharing or commercial purposes you probably would like to know a little bit more so that you can use the appropriate settings for your intended target. If this is you or you really want to understand what it all means then you will find that reading **Appendix B: Standards And Aspect Ratios** will explain and guide you through to understanding what this all means. For further information there are many options available to you. Libraries (YES, REAL BOOKS and PERIODICALS DO STILL EXIST !), online sources from many websites (Wikipedia is an excellent source for those just beginning), and of course when you want to get down to the nitty and the gritty many of the Standards are freely available. If you do go

commercial then quite often your parameters are determined by your customer!

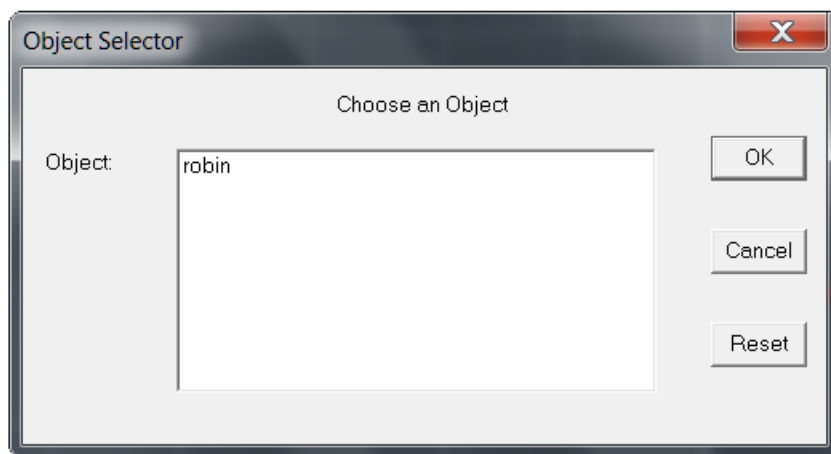
Elements of a Scene

An **element** is a part of a scene. A new scene contains two default elements, the **world** and the **camera**. The world is the global coordinate system. The location and orientation of everything in a scene is, ultimately, relative to the world. The camera is the viewpoint for the movies you make. You cannot delete either the world or the camera.


Everything that you can add to a scene, including lights, objects, figures, and targets, are also called **elements**.




Adding Objects

You add objects to a scene by selecting the **Build→Add-Object** command. This brings up a dialog showing all of the objects in your project. Select the one that you want and click OK.



It will be added to the center of the scene. This is not the actual object; you cannot edit it from within the scene editor. Instead it is a *link* to the object. But you can scale the view of it within the scene, and add multiple copies of it. The screen shot of the scene above has three instances of the seaweed object in it.

You will often find it more useful to select **world coordinates**  when adding objects to a scene. They are "dropped" into the scene at the center, at coordinates <0,0,0>, and are selected.

Make sure that the move button  is set, and then you can use the left mouse button to drag the new object around on the ground plane from within any viewpoint. The right mouse button will raised and lower it. You can also scale  and rotate  an object's size and orientation, and animate them by specifying a series of **key frames**.

Adding Figures

You add figures in the same manner with the **Build→Add-Figure** command. They are dropped at the center of the scene. You must move them to their proper place. You can animate the position of figures with key frames, and you can also attach predefined sequences of motion, as described below.

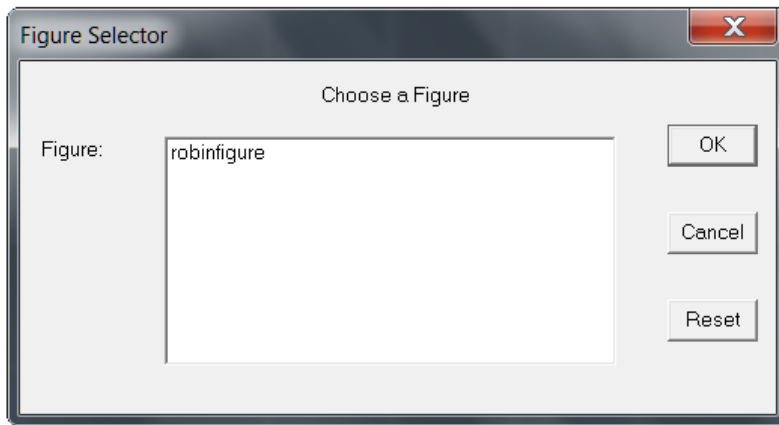
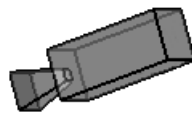


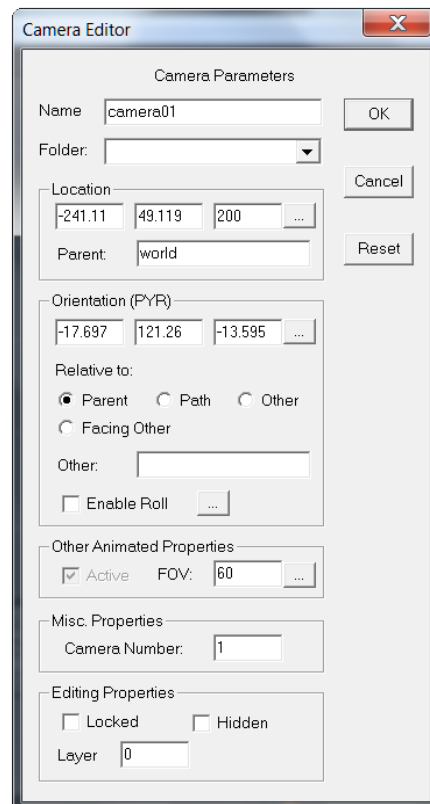
Figure Selector


The Camera



The **camera** is the reference point for the final animation. It's fully animatable. You can change its position, field of view, direction, etc. You can also animate it for any variety of pans, tilts, or dollies. You can attach a camera to another moving object for fluid tracking shots, lock a camera's direction on a moving target, and even move right through "solid" objects. To see what the camera sees, select the **View→Camera** command.

You can add additional cameras to a scene with the **Build→Add-Camera** command. The camera's **Active** controller sets which camera is currently active. Double click on a Camera to open up the **Camera Editor** to show its parameters dialog and to set the Active value or use the animation track.

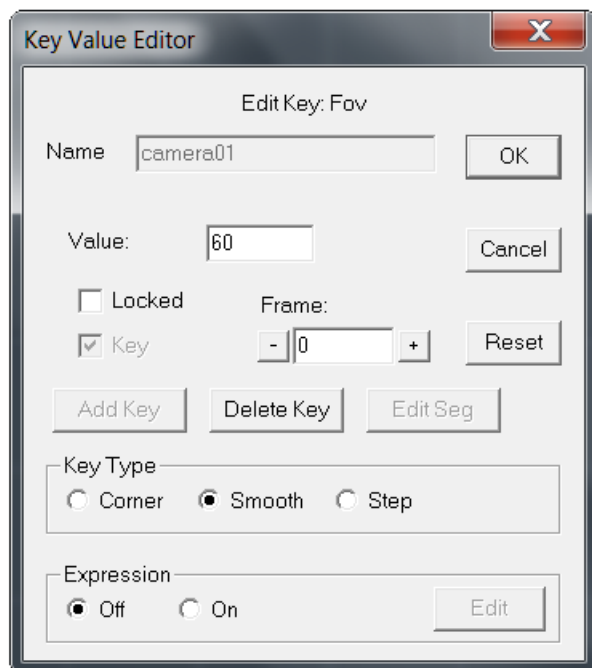


Don't forget to enable animation with the animate key  if you want your scene to use multiple cameras! Remember... If you use more than 1 camera in the timeline, then you must set the your new camera as 1 (on) and other camera to 0 (off) .

FOV

FOV or the Field Of View is the area or environment that can be observed at any given moment. The field of view is generally measured in two ways, linearly or by the angle of viewing. Linearly, it determines both the closest and farthest distances that can be observed and in Anim8or rendering it relates to the z-near and z-far clipping planes. This can be changed in the **Advanced Environment Editor** (see below).

Anim8ors' camera can also adjust its field of view by angular degrees. Angular FOV is measured in three ways: vertically, horizontally, and diagonally. It is mathematically determined by the lense and its focal length that is being used and so adjusts what the viewer is seeing. This, combined with camera distances can be useful for many varieties of stills and animated scenes. Its animation can be set in the camera **FOV Key Value Editor**. You should keep in mind that certain field of views combined with various distances may give in some circumstances unwanted results (distortions of various kinds). In rendering in Anim8or it roughly relates to the top, bottom, right, and left clipping planes of the frustum.



Lights

When you first create a scene, there are two **default lights** in it. They give the scene an overall level of illumination so that you can see what you are doing. You can't move them about or change their properties. In general you should add your own **user lights** for better results. When you do the default lights are automatically deleted.

You add lights to a scene with the **Build→Add-Light** command. This adds a light directly overhead with default properties. Double click on the light to change its color or other attribute, and drag it around the scene to change its position or direction.

Like other kinds of objects, lights can be animated fully, including their color.

There are three kinds of lights that you can use in Anim8or:




Infinite or **directional** lights cast an even light on all objects in a scene, with all of the rays parallel, like the sun. Close and far things are illuminated with the same level of light.




Local lights emit light that radiates out in all direction from a particular point in the scene, like the bulb in a lamp. Objects between the camera and the light are backlit, while those behind the light are lit from the front. The further an object is from a local light, the less illumination it receives from that light.



Spotlights are similar to local lights, but they only illuminate objects that lie within a particular cone. Things in the center are fully illuminated, while things near to the edge of the cone are gradually lit with less light.


Lights do not appear in the final scene, or in any camera view. If you want an object to appear to emit light, then you must add a light in the same location as the object. You can also toggle the visibility of lights in the other views with the show camera  button.

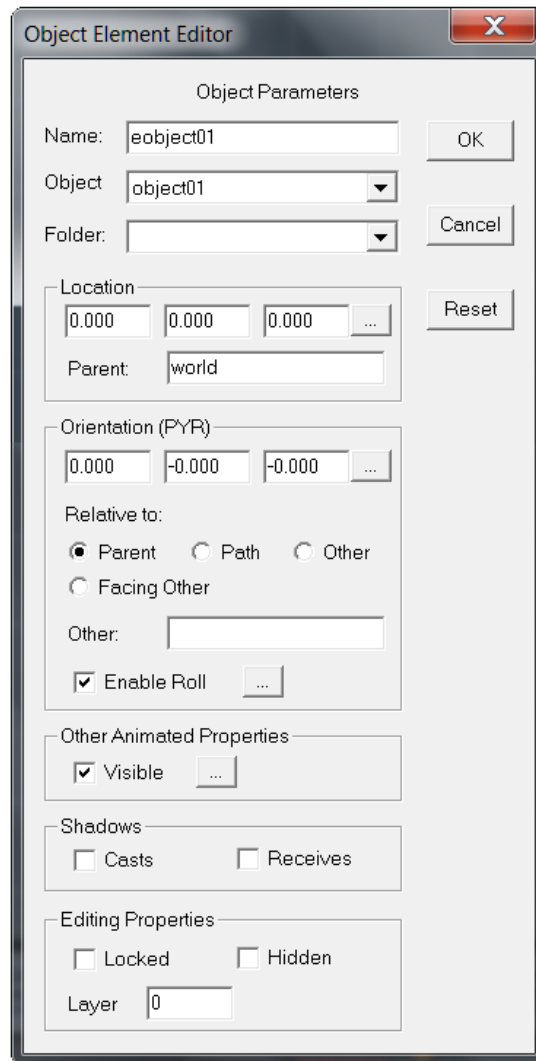
Targets

A target  is a dummy point of reference. You use them as points of interest for the camera to follow, or as a parent of a group of objects that you want to move. They are fully animatable, and do not appear in the final scene. You add a target with the **Build→Add-Target** command.

Object Properties Dialog

You can edit various properties of any kind of element by double clicking on it. This brings up a dialog similar to the **Object Element Editor** shown below.

You can animate any value that has a controller button  next to it, changing its value throughout a scene. Values with **white** backgrounds are **fixed** for the scene. You can edit them directly in this dialog. Values that are **grayed** out are **animated**. You edit their values by clicking on the adjacent controller button. If you want to animate a fixed value, click on the button to add a controller. Controllers are explained in more detail later in this chapter.

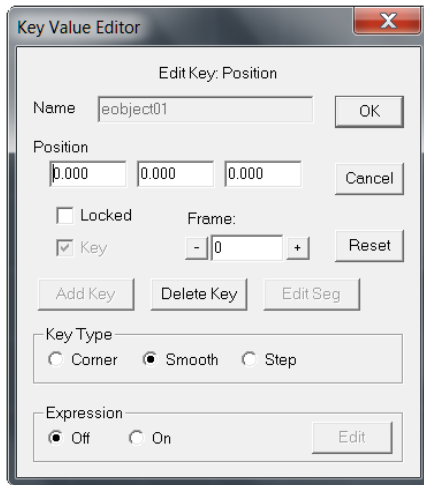


There are several common fields to all element types:

Name The element's name within this scene. Remember, an object element in a scene is only a *reference* to the actual object. There can be more than one occurrence of the same actual object in the same scene. If you change the original object in the Object editor then all of the references to it in your scenes will change as well.

Object The actual element's name. You can select any object in your current project with the drop down dialog.

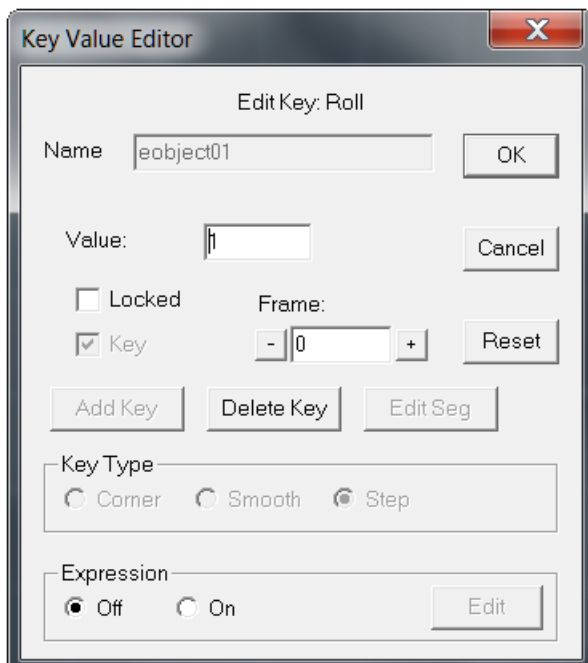
Location (Position) The location of an object in <x,y,z> coordinates is shown here. This is the location relative to its parent's location and in the coordinate system of the parent. You can animate the location.



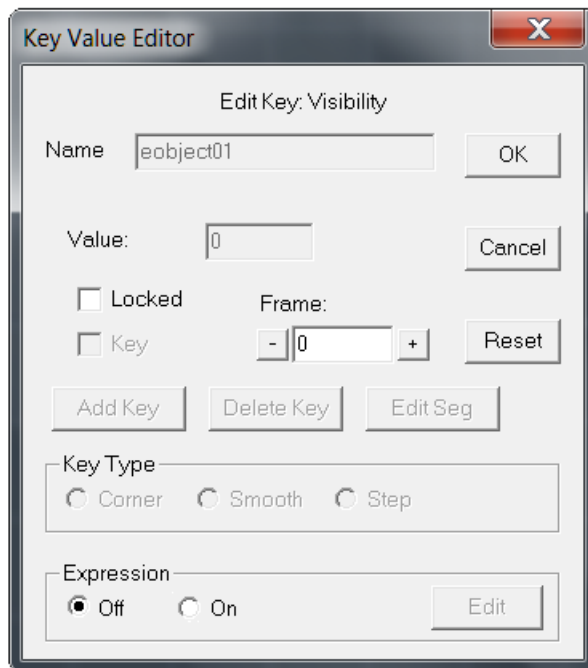
Parent Any element can have another element for its parent. Then when the parent's position changes the element's position will change as well. With a parent, the location is a relative position. To specify a parent for an element, enter the parent's name in this box. To remove an element's parent entirely, clear the box or use the default **world** parent.

Orientation This value shows you element's orientation in degrees of **pitch**, **yaw**, and **roll**. Pitch is a rotation around the X-axis, yaw the Y-axis, and roll the Z-axis. The values shown are a relative orientation to one of several coordinate systems. It can be either relative to its parent, to its path or direction of movement, to the orientation of another element, or it can always facing a particular element. You can animate the relative orientation.

Enable Roll You can disable an element's roll, keeping it "upright". You normally what to keep your camera upright so by default the camera cannot roll, but everything else can. Enable Roll can be animated.



Visible You can also animate the visibility of elements so that they appear or disappear during your scene.



Shadows Shadows add realism to your scenes but can be expensive to compute. You can decide which elements cause shadows and which ones show shadows on them. If you want an element to cast a shadow you must enable it with the **casts** checkbox. If you want it to show shadows check the **receives** checkbox.



Editing Properties The Editing Properties values are useful when editing your scene. You can **hide** an element by checking the "hide" box or with the **Edit→Hide [h]** command. This is useful to allow you to see what's behind something, or to simplify a scene for faster drawing. To show a hidden element, either uncheck the "hide" box, or select the **Edit→Show-All [H]** command.

You can **lock** an element by checking the "locked" box. This prevents any changes to any of its properties within the scene, such as location or orientation. This does not mean that it cannot move. If it is initially animated, it will stay that way. Locking only prevents you from accidentally changing that animation.

Note: This does not mean that the element cannot be changed in any way. For example, if it has a parent, then when the parent's location changes, so will the element's position, since its definition of location is relative to the parents.

You can also assign a **layer** to each element for use with the usual layer controls.

These values are only present for certain kinds of elements:

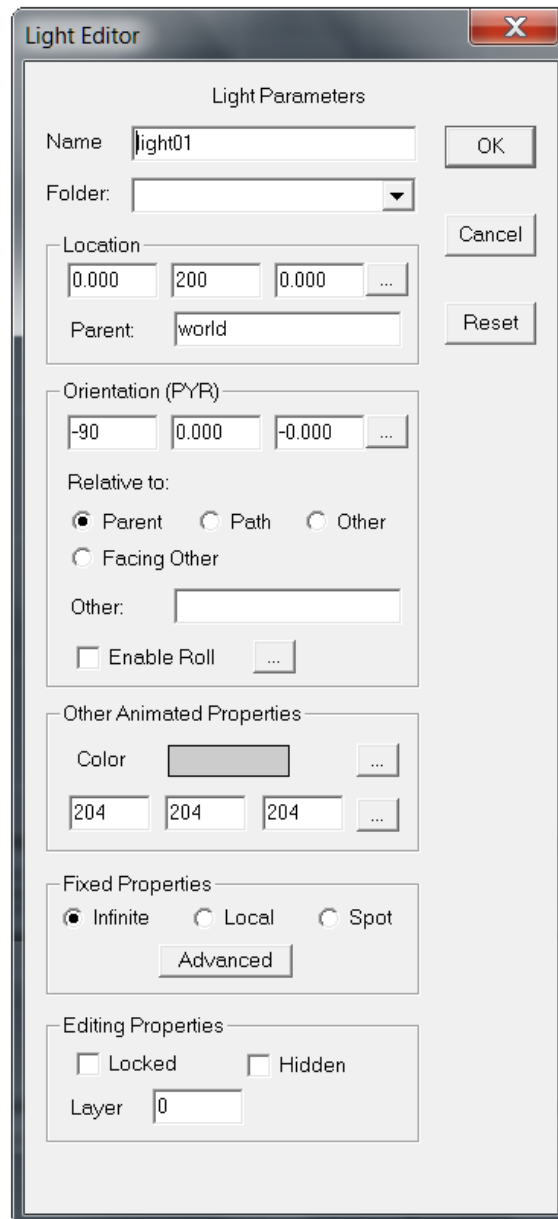
Scale An element's size can be animated. However you do not use a dialog to do it. Instead you simply select scaling , enable animation  and change the size.

FOV You can set and animate the camera's **fov** or **field of view**, the angle of the width of the image the camera shows on the screen. Slowly increasing or decreasing the fov zooms the camera out for a wider view or inner for a tighter view. This value is in degrees.

Color You can also animate the color or brightness of a light.

Light Properties Dialog

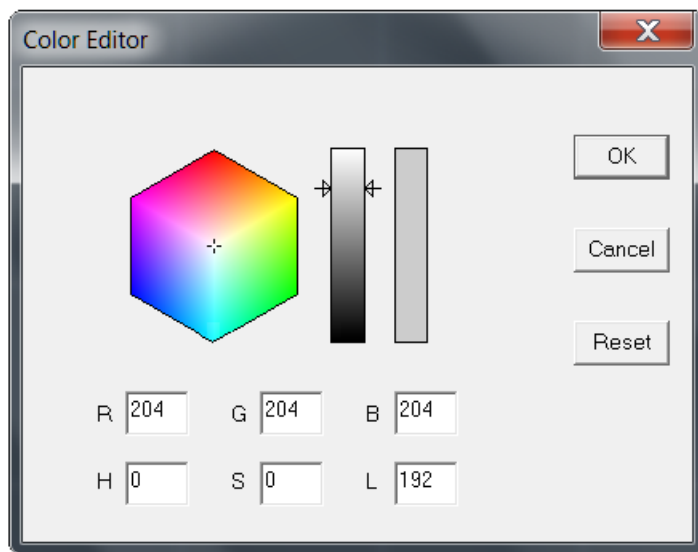
Lights have some unique settings that you can use. You access these from the **Light Editor** dialog by double clicking on the light.



You change the color of a light by entering the numeric value directly, or by clicking on the



next to the color sample.



You can animate its color with the  button next to the numeric color values.

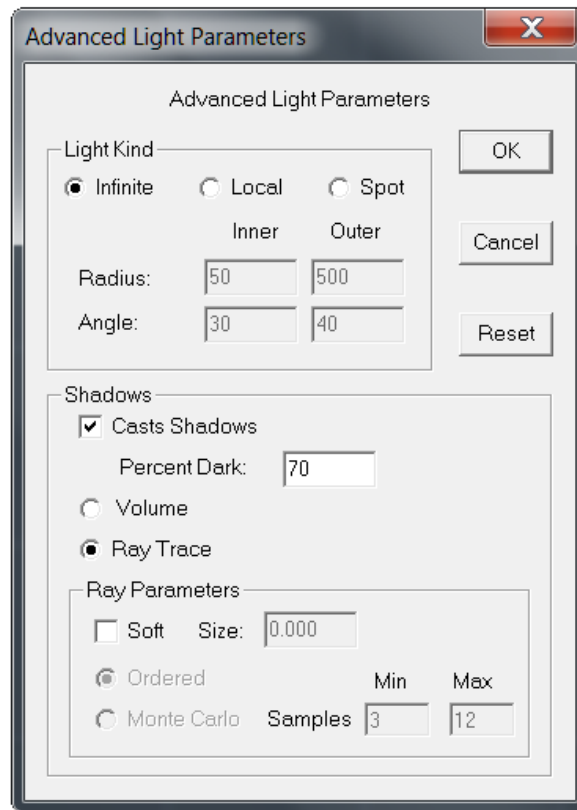


You can change your light into an infinite, local, or spot light in the **Fixed Properties** area. To access more advanced settings click on the  button.

Advanced Light Properties Dialog

You can set several important lighting properties in the Advanced Light Parameters dialog. For local and spotlights you can set limit the light's **range**. These lights have a steady brightness out to a certain distance called the **inner radius**. Beyond that distance the light's strength gradually diminishes until at the **outer radius** and beyond the light has no effect at all.

Spotlights are basically cones of light projecting from a point. You can set the angle of this cone as well as how sharply it falls off to the edge. Within the **inner angle** the spotlight has full brightness. Outside that angle the light's illumination gradually weakens until the **outer angle** where it had no further effect.



Shadow Properties of Lights

By default lights do not cast shadows. You enable shadow casting by checking a light's **casts shadows** checkbox.

Shadows in the real world are not totally dark. Some light is scattered off of nearby objects into these areas. You can illuminate the shadows in Anim8or as well with the **percent dark** parameter. This only allows diffuse light into shadows so that there are no shiny highlights, which would appear unrealistic.

There are two main kinds of shadows, **volume** and **ray traced**. You select which kind you want you light to cast in this dialog. Ray traced shadows can cast soft edges. You control how soft the shadow is with the **size** parameter. For infinite lights this is the angle that the shadow spreads from the edge of an object. For local and spotlights this is the physical size of the light. Objects closer to these lights will cast softer shadows than those further away.

You can set the minimum and maximum number of samples used to calculate soft shadows in this dialog as well.

The section on Shadows has more information on using shadows.

Shadows

Shadows are a very important part of an image. They can also consume an enormous amount of computer time. It's important that you understand how they work or you may spend a lot of unnecessary time waiting for your images to render.

Shadows are only visible in rendered images made from the Scene editor. They are only cast from light that you add yourself. And they are only cast by object that you choose. To enable shadows you need to:

- Use the Scene Editor,
- Add your own light and enable the **casts shadows** in the advanced properties dialog,
- Enable **casts shadows** and **receives shadows** on the element of your scene that you want to show shadows, and
- Render an image or a movie

Volume Shadows

Volume shadows are usually much faster to render than ray traced shadows but they have some limitations. They have hard edges and you cannot use them to cast soft shadows. This tends to make your images look unrealistic.

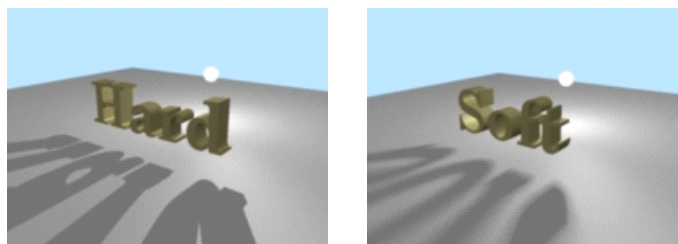
A bigger limitation is that you cannot use volume shadows to cast shadows from within an enclosed object such as a jack-o-lantern. While this is not a problem in most scenes it can cause you some unwanted "surprises" if one of your lights moves into an area enclosed by a single object. You can work around this problem by splitting the object into two or more pieces, or by using a ray-traced shadow for the offending light.

Ray Traced Shadows

Ray traced shadows are the most versatile. They can also consume a very substantial amount of time to render.

Ray traced shadows can be either **hard** or **soft**. Hard shadows have a very distinct boundary between what is inside the shadow and what is outside. Your shadow from the sun on the ground is an example of a hard shadow.

Soft shadows don't have a distinct boundary. Instead the amount of light gradually lessens until the shadow reaches its maximum darkness. The shadow cast by a florescent light is an example of a soft shadow. Such shadows are soft because the apparent size of the light casting them is large. As you can see below soft shadows can be much more realistic.




Soft shadows are costly to computer. Many individual shadow samples are used to make a soft shadow. Anim8or therefore limits soft shadows to anti-aliased images where multiple image samples are used for each pixel. High quality soft shadows often require even more samples than Anim8or uses for anti-aliasing. To limit the total cost of these computations, Anim8or adapts the


number of samples to the image as it is being rendered. If there is nothing "interesting" in an area then only a few samples are taken. If a shadow edge is detected, however, more samples are taken to improve the image. You can set the minimum and maximum number of samples used for each light in the Advanced Light Properties dialog.

In general, you will need to set the number of samples higher for larger sized lights that cast soft shadows. You will also have to increase the number when a light is very close to an object since this will magnify its shadow's soft part.

Environment Settings

You can use the Environment Settings dialog to control several things in your scenes. You can add a **background image**. It can be either a fixed image or a **panorama** that moves with the camera pans. You can also add **fog** effects. You open this with the **Scene→Properties→Environment** command.

In the **background** area you can set the background for your scene several ways. By default it is a solid color. You can change the color with the  button next to the color sample.

You can also use a fixed image for the background. To do this select the "image" button and load a background image from a file with the  button.

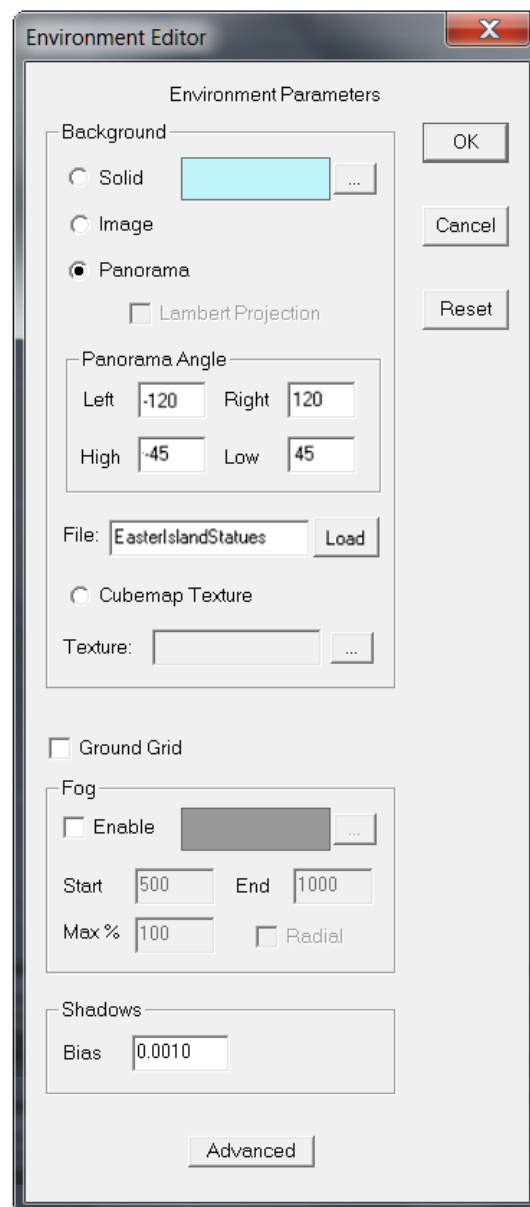
You can also use an image as a **panorama**. The image is wrapped around your scene in a big arc. When the camera moves it will pan across the background. The diagram below shows how this works.

You set the area that your background fills in the Panorama Angle area. You have to be careful not to aim the camera past the edge of the panorama or the background color will show.

You can also use full 360 degree VR backgrounds. For these you need to set the angle limits to -180 to 180, and -90 to 90. There are two common projections for VR images, **linear** and **Lambert**. Lambert is used more often for scientific work because it has approximately the same solid angle for each pixel. If the background is distorted when the camera is pointing straight up or down try changing the Lambert check-box.

You can remove the default **ground grid** from your scene by un-checking the Ground Grid box.

Distant **fog** can add realism to your scenes. It is useful for underwater shots as well as distant outdoor views. You enable for by checking the box at the top of the fog section of the Environment dialog. You can set the color, the distance from the camera at which fog begins to appear, and the distance beyond which it has maximum saturation. You can also limit the maximum effect it has so that things don't disappear entirely in the distance.

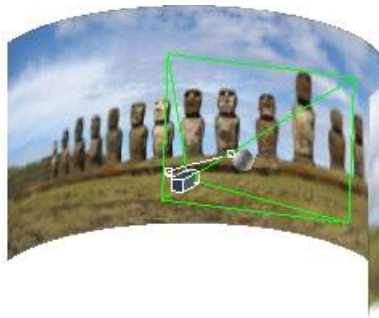


Shadow **bias** is an advanced technique used to hide ugly rendering artifacts that can appear when

shadows are used. These artifacts are caused because the lighting computations assume that your models have smoothly curved surfaces. In reality they are made of flat faces. Shadow computations are based on these flat faces and don't always get along with the lighting computations.

To overcome this discrepancy, a shadow bias is used to push shadows slightly back away from the viewer. Then everything looks fine.

You normally shouldn't have to change the shadow bias.

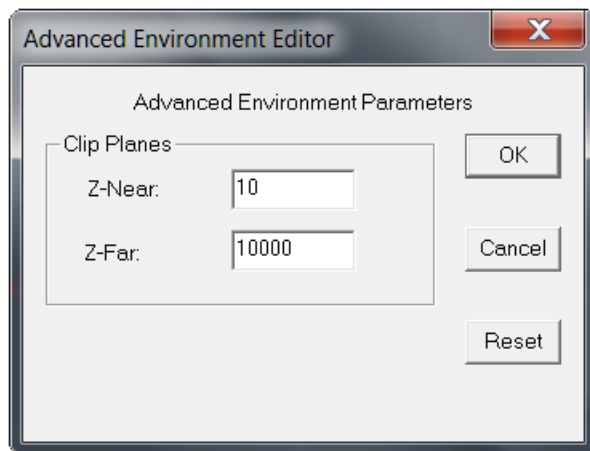


Panorama Environmental Image

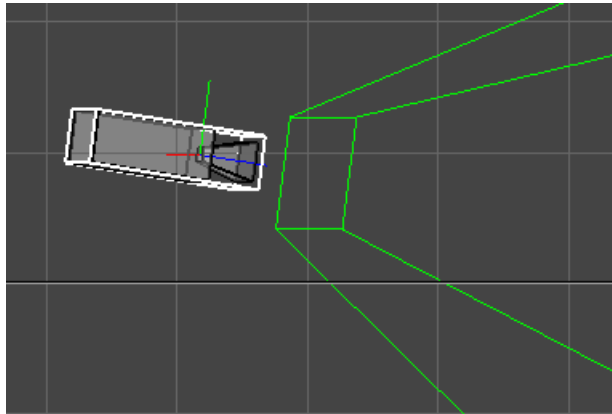
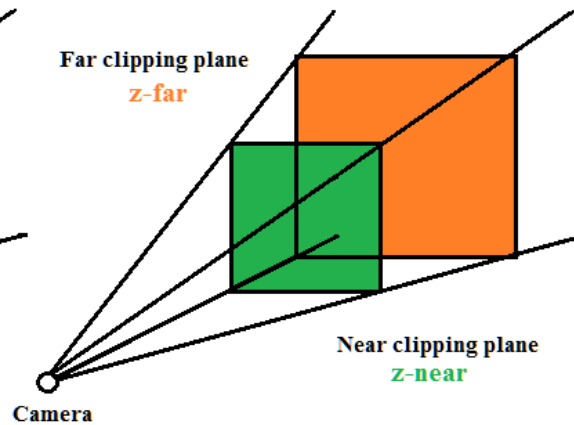
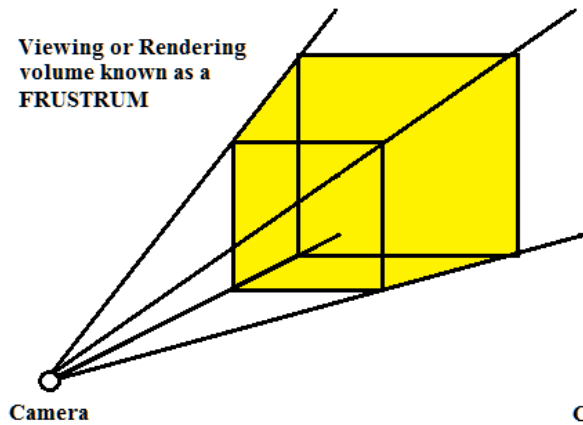
Note: Backgrounds only appear in the Camera view.

Advanced Environment Editor

Sometimes you will find as you are moving around or rendering your model or scene that portions seem to be cut off or missing. You may need to adjust your clipping planes and this can be done under the button. This can also be accessed in any mode by going to **View→Z Limits**.



In order to save computational time, whether it is for an image, animation, or in 3D gaming the scene is clipped so that only the parts of that scene in the frustrum are rendered. This means that **ONLY** what is inside the viewing volume is considered. If your scene is quite large it may extend outside the boundaries and so are ignored and not rendered. Adjusting your z-near and z-far parameters should fix this problem. Just remember that if your clipping planes are quite distant that the computational time will increase. Otherwise there should be no problem.






Camera and Frustrum in scene editor

The Time Track

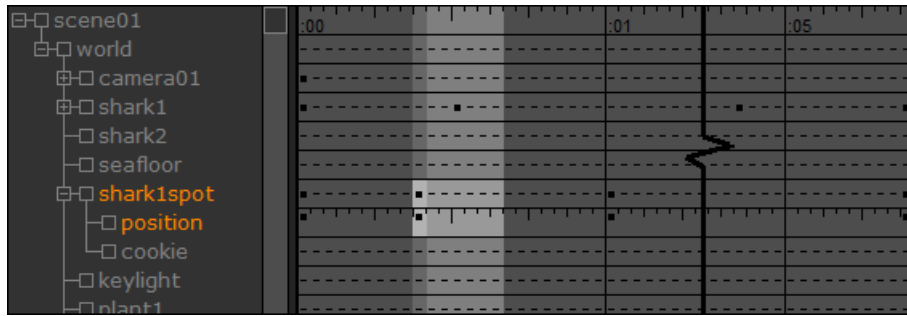
At the bottom of the window is the scene's **time track**. If it isn't visible on the screen, select the **Options**→**Track-Window** command:




If you prefer, the Time Track now has the option of becoming a floating window in both the Sequence and Scene modes. You can find this under **Options**→**Float Time Track**. On the left is the scene's title. It is the top of a tree description of the elements that make up your scene. If you click on the "open" box  you can expand the window to show the objects contained in the scene. You can change height of the expanded track window by dragging the horizontal resize bar  up or down, and the width of the name field by dragging the vertical resize bar  to the right or left.

The right part of the track is a time line for the scene. If you click on it, the display will switch to the time you selected. Each tick mark represents one frame. Seconds are marked with full vertical lines. Normally there are 24 frames/second but you can change this to other frame rates with the **View**→**Preferences** dialog. The current frame is highlighted in dark gray.

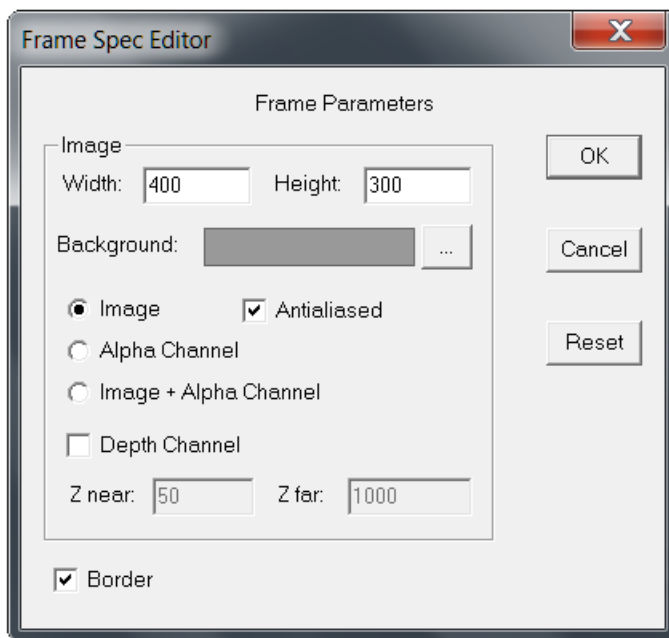
You can click and drag the mouse it will select a range of frames. Use Shift-Click to select a new frame range while still keeping the same current frame.



When expanded, the left part of the track shows the objects in your scene. There is a lot of useful information here. You can see what an object's parent is, what properties have been animated, which bones can be rotated, and such. In the view above the highlighted object *shark1spot's* parent is the world. Its location is animated and it has a child object called *cookie*. If you see an "open" box  by an item then it has more information that you can display by clicking on the box.

You can select and edit items shown in the name section of the track by clicking and double clicking on them. Mouse buttons have their normal meanings with the right one adding to the selection and the middle one subtracting. Double clicking is how you can edit the numeric value of animated properties like an element scale that does not appear in a dialog.

Previewing and Saving Images



You can preview a software rendered image of any view with the **File→Preview** command. (This command is not used in the Anim8or 1.1318 build at this time. Use the following information to preview your image ; you do NOT have to be in the Scene Editor to use this.). You can set the size and background color used to make the image, and can choose to make it anti-aliased. You can also use the **Render→Render-Image** command to do this in the Scene Editor. Explanation of how to render a preview or final and save your image is found in **Chapter 12 Rendering**.



Scene Animation


You animate the elements of your scene by setting their location, color and size etc. for a few **key frames**. The computer supplies the remaining values between these keys. You can take more control of how things animate by adding more keys. Besides their basic value some keys have additional properties that you can edit for further control.


A **segment** is a set of values for the range of frames between two successive keys. You can change how fast things move between key values by editing the properties of segments.

You can also use a **script** or **expression** to animate elements. A script is a simple program that computes the a particular value from the frame number or time in seconds, or from the properties of other elements.



Animation with Key Frames

You can animate your elements by simply moving them on the screen. First enable animation by clicking on the animate  [K] button. Its background turns bright blue  indicating that you are now animating. Change to a new frame by clicking on the frame in the time track. Then select an element and move it to a new location. Anim8or will add **key frames** for your element's location for these two frames and will smoothly fill in the locations for all of the frames between the keys. When you press the play button

 [cr] your element will now move!


You will also notice that a **spline path** has been added to the scene. This is the element's path through space. You can add more points at different frames to build complex motion. You can also edit the shape of the splines by clicking on the edit spline  [p] button on the toolbar.

Editing objects positions is described in more detail later on. But there are a few things to note:

- When the animate button is off  any change that you make will apply to the entire scene. If you move the camera, for example, it will be moved the same amount it each frame. When the animate button is on  changes that you make will only add or change keys for the current frame. Nearby frames will be smoothly changed to reflect these changes.
- When you move or rotate an element that is a child of another element, then the change will be *relative to its parent*. The same is true when you rotate an element that is facing its path of movement. The change is relative.

You can edit individual key's values by double clicking on the key in the time track.

Animating with Inverse Kinematics

The Inverse Kinematics tool  [I] allows you to set multiple keys simultaneously. As

described in the Sequences chapter, you select an IK chain and drag the end effector to make your desired pose. However the Scene editor has a much more powerful capability. By locking the location of end effectors, when you move a figure the locked IK chains will adjust to stay as close to the locked position as possible. Double clicking on an end effector displays a dialog for locking and unlocking them.



Note: the locked property is animatable so you can change the value through-out a scene.

Forum Notes:

Difference between Lock and Hold

(forum discussion between Steve and member johnar, October 16, 2016)

johnar: "There is a bit of a misunderstanding between 'lock' and 'hold', which are essentially the same, but just under a different name.

Traditionally, the correct word is 'hold'

Anim8or has keys, just as any other animation program, and a 'lock' key is really just a 'hold' key anyway, which will 'lock, or hold' that position until a different key is set, in which case it has become 'unheld, or unlocked'.

Same with rotation/(orientation) keys.

This is how anim8or already works, so i'm pretty sure the usual key method would be no different for 'holding/(locking)' an end effector into place."

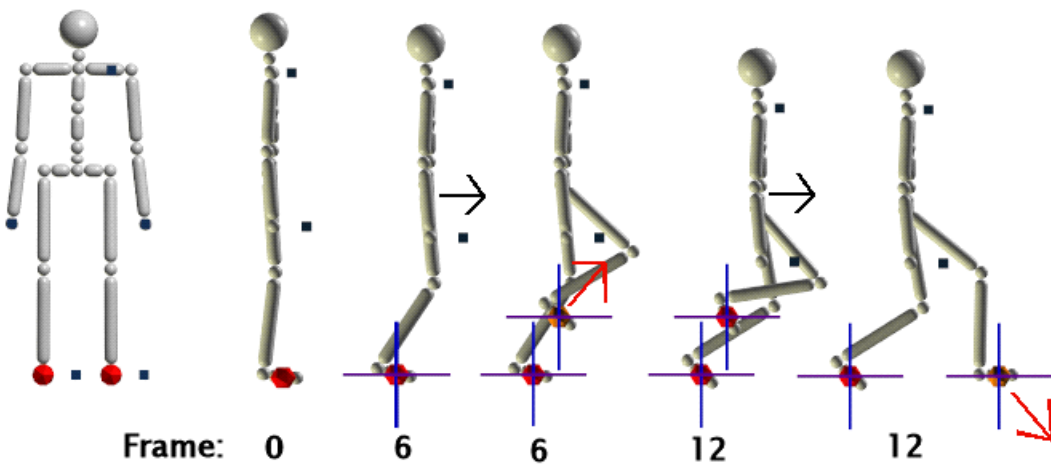
Steve: "I think the term "locked" is a bit misleading; I should pick a better one. What "locking" an end effector does is actually enable IK. When the "locked" effectors are animated (I'm working on this now) they will move in a manner similar to any other key-animated point, along with the position of the figure. After these values have been set, the IK system will position the joints in the IK chains for all "locked" effectors.

If an effector isn't moved between two keys, it is locked in place, hence the term locked. The spline interpolation needs to be slightly different for effectors or they won't be held exactly in the same place between two keys with the same value. Instead there is a bit of overshoot caused by the so called "lowest energy" shape that they normally assume.

A better term might be "active" or "enabled" or "". Any ideas?"

Inverse Kinematic Walk Cycle

Below is a simple human figure with IK chains attached to the arms, legs and back. To make the character walk, start in frame 0 and lock both leg chains. The end effectors are shown in red to indicate that they are locked.



Creating a walk cycle using IK

Change to the Right view and advance to frame 6. Move the character to the right and slightly down. Since both feet are anchored to the ground the knees will bend as shown. Now drag the right foot up and over so that it's under the middle of the character. This is now the key pose for frame 6.

Advance to frame 12 and move the character to the right and down a bit more, then drag the right foot out in front so that it is flat on the ground. Repeat these steps for frames 18 and 24 but moving the left foot and you've created a walk cycle.

You can make adjustments to individual bones if the initial angles aren't what you want. The animation is often smoother if you do this on the same frame that you make keys using IK, but it's not required.

Animation Figures with Sequences

You can also animate a figure by adding an entire sequence of key positions from a sequence. If you make a sequence that is one cycle of a walk you can then add to a scene several times in succession to make a longer walk. Sequences appear are shown in the track bar by <——>. In the track below there is a 12-frame sequence shown for three joints.



Sequences in a scene are linked to the sequence editor. Any changes that you make to a sequence will automatically appear in your scenes. For this reason you cannot modify keys that are part of a sequence in the scene editor. If you try Anim8or will ask if you want to convert that particular reference to the sequence into individual keys and continue. Converting a sequence here will not alter the original sequence, and the link between the sequence and the scene will be lost so that further editing of the sequence will not change these keys in your scene.

Animating with Expressions

Some things can be tedious to animate an action using keys. For example, if you want a light to blink on for a short time every second you could add two keys for every second in your scene. One would set the light's color to its normal "on" value and the other would set it to black so that it would appear off. This is not only time consuming but if you make the Scene longer you must add more keys to the light or it would stop blinking before the end. If you decided that the light should be on for a longer time each flash you would need to move half of the keys.

Anim8or supports a the third way to animate elements, expressions, which makes tasks like this very simple. An expression is basically a small computer program written in Anim8or's scripting language. Each time you change frames Anim8or evaluates these programs with the time or frame number updated to their current value. The expressions then choose a value that's appropriate for that time.

This is a sample controller expression:

```
$color = (0.8, 0.1, 0.1)*(fract(time) < 0.2);
```

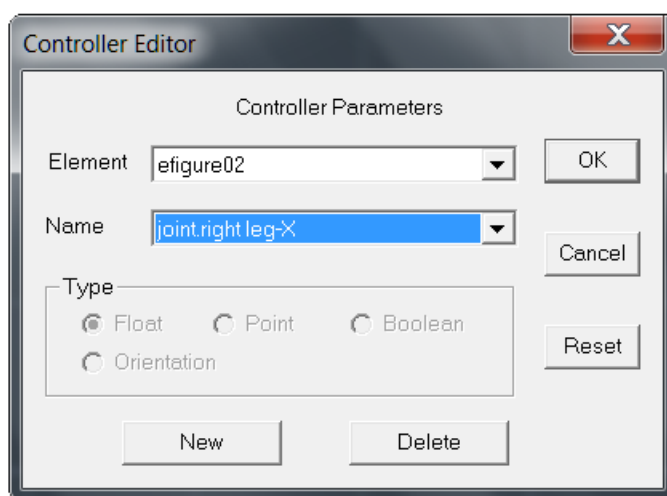
Used for the color of a light it would cause it to blinks red one a second. Chapter 10 Scripts describes the rules for writing your own expressions.

Controllers

Controllers manage key values for animation in the scene editor. When you move, rotate, or animate any aspect of an object that uses a key, a controller is created that holds that key. Each one manages only one aspect of the scene. For example one might control the location of an object, and another the color of a light. They contain all the keys regarding that value and any related data.

If an aspect of an object is constant for the scene and not animated then it will not have a controller. The one exception is for bones. They always have a controller for each axis in each joint that can bend. This makes it easier for you to find which bones can move when you are posing a figure.

You can access the Controller Editor by going to **Edit**→ **Controllers**.



Kinds of Controllers

There are several kinds of controllers. Each uses a different kind of data, or computes in-between values in a different manner, depending on the needs of the property that is animating.

Float This is an ordinary number like 10.5 or -150. Like most controllers, in-between values are smoothly interpreted.

Point3 Point3s return 3 numbers per frame. Each value is smoothly interpreted without regards to the others. You use these for positions and RGB colors.

Orientation An orientation controller returns a rotation per frame. They can set the orientation of an object or can be used for a rotational direction. When you edit these values in a dialog you see the PYR (pitch, yaw, roll) equivalent to the orientation.

Orientation is more complex than it may seem. Interpreting angles smoothly will not result in smooth rotation of your models. The order in which pitch, yaw and roll are applied is also important. Anim8or uses PYR because by applying roll last you can set it to 0 and your elements will stay "upright".

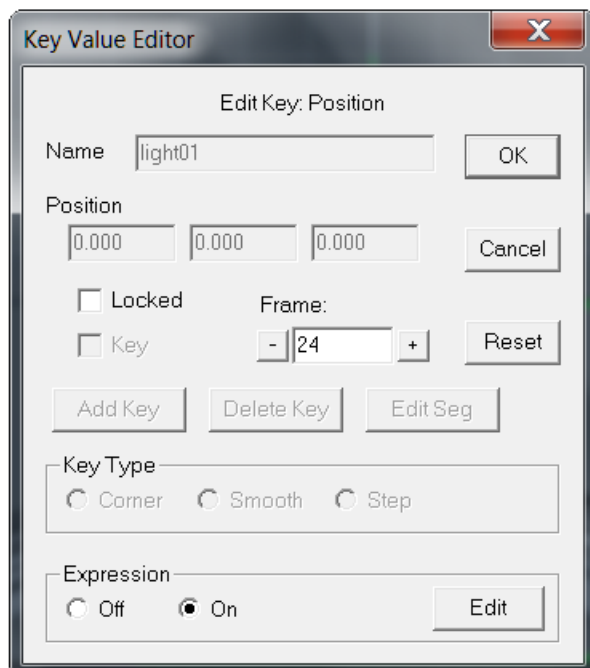
It's often easier for you to just ignore the numeric values for orientation. Simply orient things on the screen and leave the rest to the computer.

Boolean Boolean controllers return a "yes" or "no" value. You would use one for an object's visibility if you wanted it to suddenly disappear. There is no "maybe" value for a Boolean.

Editing Keys

You edit a key by double clicking on its key indicator ■ in the track, or by clicking on the

☰ button for its controller in an element's properties dialog.




You can edit a location key's x, y and z values in a key location dialog, and add or delete keys. There are similar dialogs for the other kinds of keys. Entering a frame number lets you move to any frame in your scene, and clicking on the + and - buttons move to the next and previous frames. You can see but not change non-key frame values. The computer supplies them.

The **Edit Seg** button allows you to edit the segment next to the current frame.

There are three kinds of keys: **corner**, **smooth** and **step**. Smooth matches the direction and speed that the value changes as it passes through the key. It is what you usually want when you are animating and is the default. Corner keys can have different incoming and outgoing speeds and directions. You would use this where a bouncing ball strikes the ground. Step keys are like corner keys but allows the value to change abruptly. You would use this to make a character suddenly change positions.

To use expressions for a controller instead of keys, click the "on" button in the Expression area.

Editing Expressions

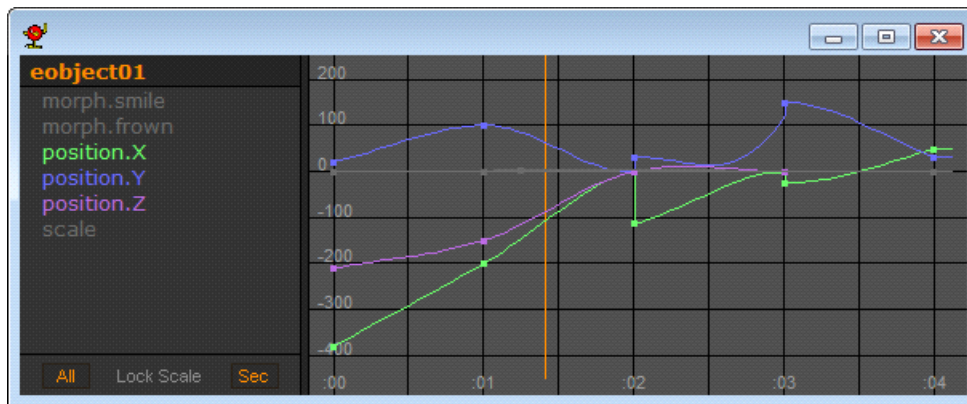
You edit a controller expression by first opening its Key Position Editor dialog, enabling expressions with the "on" radio button, and then clicking the  button in the expression area. This displays a window in which you can enter and edit the expression.




See **Chapter 10 Scripts** for more details on writing expressions.


Graph Editor




You can also edit the controller's for a scene's elements with the **Graph Editor**. It's displayed in a separate window that you open with the **Options→Graph-Editor [Ctrl-P]** command. The left panel shows a selected element's name and the names of its controllers. The right panel shows a graph of the controllers' values.



Click on the name of a controller in the left panel to select it. As usual the left mouse button selects a single value, the right button adds to the current selection, and the middle button deselects. Double clicking on a controller's name will display its key editor dialog described above.

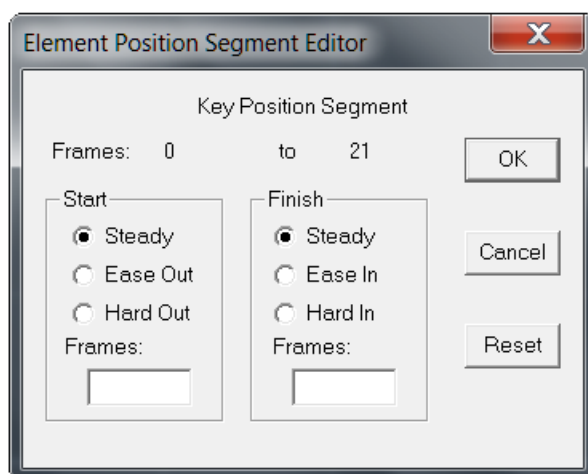
Once selected you can edit a controller's keys in the right panel graph view. Click on a key and manipulator handles  will appear. Click and drag your mouse up or down on the center handle or the lines connecting the handles to adjust a key's value. Click-drag on the end handles to change the rate that a key's value changes. The scene will be updated as you adjust the graph so you can see how much you want to adjust a value.

Handles for step keys  are shown in two colors. The incoming value is in yellow and the outgoing value in white.

If the graph is too cluttered to see a particular controller you can hide deselected controller's values by clicking the All button  off. You can lock the vertical scale to its current values with the Lock Scale button , and toggle the labels on the horizontal axis between frames and seconds with the Sec button .

Orientation controllers may not behave as you'd first expect for several reasons. They wrap at +180 and -180 degrees so the graph can look strange as it crosses from the top to the bottom. Also changing one axis's value can affect the other axis' values. Don't worry, that's just how orientations look when show using the familiar pitch, yaw, and roll of Euler angles.

Editing Segments



By default values vary at a consistent rate between keys. But this is not always what you want. How quickly a character reaches full speed when starting to move, or how long it takes to slow down when stopping, are important elements in animation. We normally think of small, lightweight objects as accelerating rapidly to full speed, but large or heavy ones as taking a long time to build up steam. In animation terms something is said to ease out and ease in if it is slow to start and stop.

By editing the segment's properties you can change the speed a value changes in several ways. You can set the number of frames that it takes an object to change speeds, size, color, etc. at the start or end of a segment with the ease in and ease out settings. You can also select the opposite: hard out and hard in where you might want an object to accelerate at the end just before it splats into a wall to increase the visual effect.

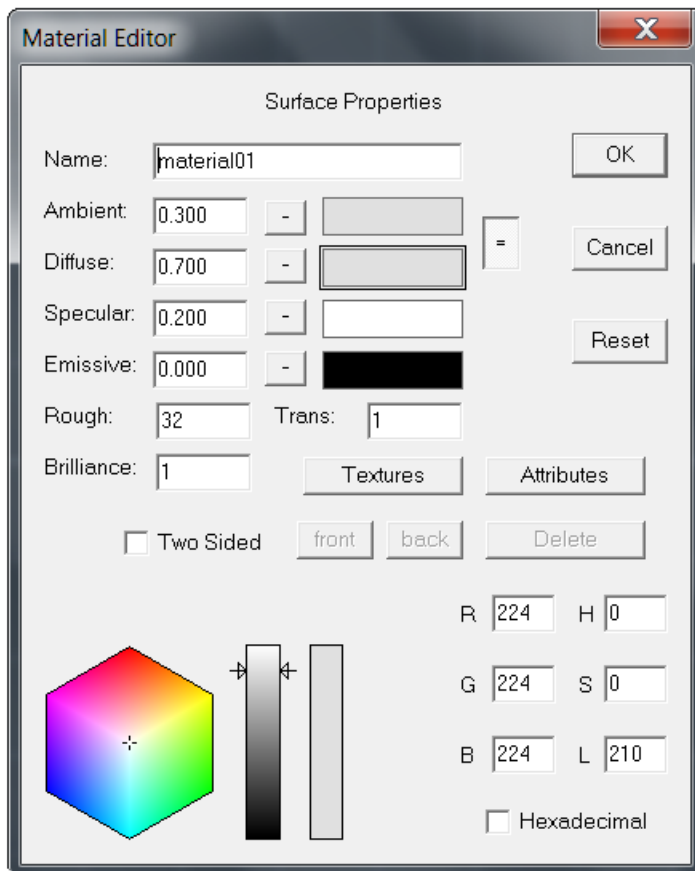
Materials

Every visible surface has a material. By changing their various properties, you set the look of your figures and objects. Materials provide a richness and detail that adds considerably to the realism of a scene. You can make them shiny, dull, transparent, and opaque. You can texture a surface with an image used as its material's color, or to lighten or darken its basic color. You can also use a texture to control the transparency.

You can't really design the look of a scene with materials alone. Its ultimate appearance also depends heavily on the lighting you choose.

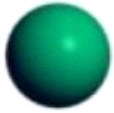
Material Editor


You use the basic material editor is whenever you need to design the color, texture, transparency, and other visual properties of an object's appearance. You open the material editor either by double clicking on a material in the material toolbar, or from a specific object's editor dialog. An example is shown below.





Some of the entries should be fairly obvious to most of you. Others can probably use a bit of explanation, so there's a short description of them in the following table. You can see how each property affects the look of the sample material shown in the ball below. Its parameters are all set


to the defaults shown above with just a single value changed for each one. Ambient and diffuse are not locked together. The base colors of the material shown to the right are:



Ambient: R=0, G=92, B=224: 

Diffuse: R=0, G=255, B=128: 

Specular: R=255, G=255, B=255: 

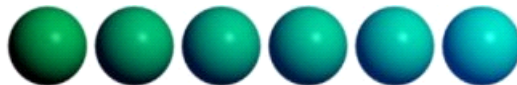
Emissive: R=255, G=0, B=0: 

Here is a description of the values in the Material dialog:

Name The **name** of the material. This field can be left as the default "material00", but it is a good idea to give each material a useful name.

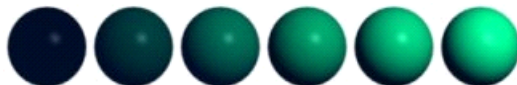
The **ambient** component is the color of a material in a shadow. Normally you want it to be the same as the diffuse component. In fact this is so common that there is a button that will lock all ambient and diffuse values together so you don't have update them both to make changes.

Ambient The number to the left is how much ambient component adds to the final color. The normal range for the ambient weight is from 0.0 to 1.0. The row of materials below have an ambient value ranging from 0 on the left to 1 on the right:



The **diffuse** component is what you would normally call the "color" of a material. It is combined with the amount and color of light illuminating your model and added to the final color.

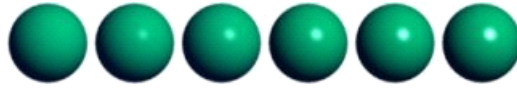
Diffuse The normal range for the diffuse weight is from 0.0 to 1.0. These samples vary the diffuse value from 0 to 1:



The **specular** component is part of the "shininess" contribution. You normally set this color to white to reflect the light's color. For metallic surfaces you should change the color to something closer to the diffuse color.

Specular

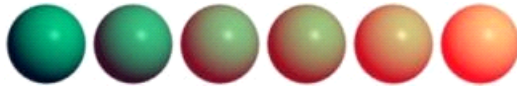
The normal range for the specular weight is 0.0 to 1.0. These samples vary the specular value from 0 to 1:



The **emissive** component represents light generated by a material. It is not affected by lighting. You use this for things like lava and eyes that glow in the dark.

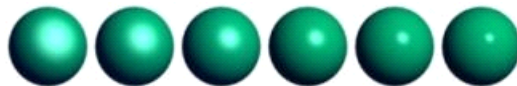
Emissive

The normal range for the emissive weight is 0.0 to 1.0. The default value is 0.0 to prevent glowing. These samples vary the emissive value from 0 to 1:



The **roughness** of the material. A higher value makes the surface look shinier. It is tempered by the value of the Specular component. Values range from 1.0 (not at all shiny) to 100. These samples vary the value from 2 to 64 by multiples of 2, and have the specular weight increased from 0.2 to 0.6 to show the changes better:

Rough

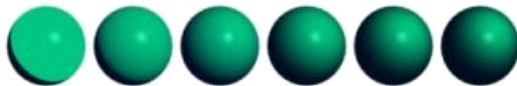


Trans*

The **transparency** of the material. Actually, it's the opacity of the material. 1.0 is fully visible, while 0.0 is completely transparent.

Brilliance*

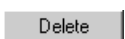
The **brilliance** factor. This setting changes the appearance of the diffuse component. Normal objects have a value of 1.0. If you increase it by a small amount to 1.5 or 2.0, the material takes on a sort of metallic sheen, or for brighter colors, a deeper, richer appearance. A value less than one flattens the look of the material. Here the value of brilliance changes from 0 to 2.5 in steps of 0.5:



The **texture** button opens the general texture dialog for this material. You may use several textures on a single material, changing the diffuse color, transparency, emissive color, and more. You can also apply a bump map texture. See the Texture Dialog section for more details.

Two Sided

You can select this item, to give the material a different material for it front and back sized. The front and back buttons will show which side's properties are currently showing.



This button **deletes** a material. You will be notified if it is currently in use and given a chance to cancel the delete.



These four color patches show the current value for the ambient, diffuse, specular, and emissive colors. The currently active one is outlined and the lower part of the window shows its numeric value. You can change the active color to a different patch by clicking on it.







These buttons show you if a particular component uses a texture. If the button has a “T” on it then it uses a texture. If not then it doesn't. You can click on a button to set a particular texture.

***Note:** Transparency and some more advanced texture modes may only affects rendered images. Its final effect can only be partially shown in interactive sessions. Some graphics cards can show more than others, depending on which features they support. You should render a test image to see the final look of your models

A few simple material images and their settings are shown in the table below. Notice the values for Pearl are out of the normal range. The ambient weight is more than 1, the diffuse is negative (which means lights actually darken a surface), and the specular value is way more than 1.

Don't be afraid to experiment!

Sample	Material	Ambient	Diffuse	Specular	Rough	Brill.	R/G/B
	Copper	0.3	0.7	0.6	6	1.8	228/123/87
	Rubber	0.3	0.7	0.0	N/A	1	3/139/251
	Brass	0.3	0.7	0.7	8	2	228/187/34
	Glass	0.3	0.7	0.7	32	1	199/227/208



Plastic

0.3

0.9

0.9

32

1

0/19/252



Pearl

1.5

-0.5

2

99

1

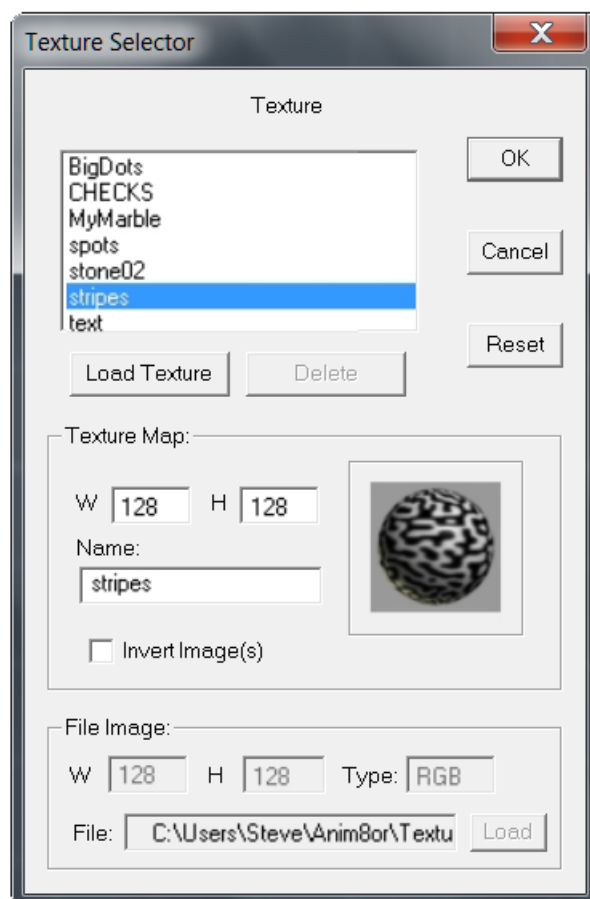
255/138/138

Texture Selector

The texture selector allows you to load, view, and manage the textures that you use in your animations. You open this dialog from within the material editor with one of the 4 buttons:



and . An example is shown below:



You can change the name of the texture in the Name field of the Texture Map area and can view the size of the image Anim8or uses to store it.

You can also invert the image in the texture by checking the **Invert Image(s)** check box.


The File Image area gives information about the file containing the texture. The Type field can be either RGB for normal images, or RGBA for .gif files with a transparent color. If your texture is type RGBA then you have more flexibility in what you can do with it.

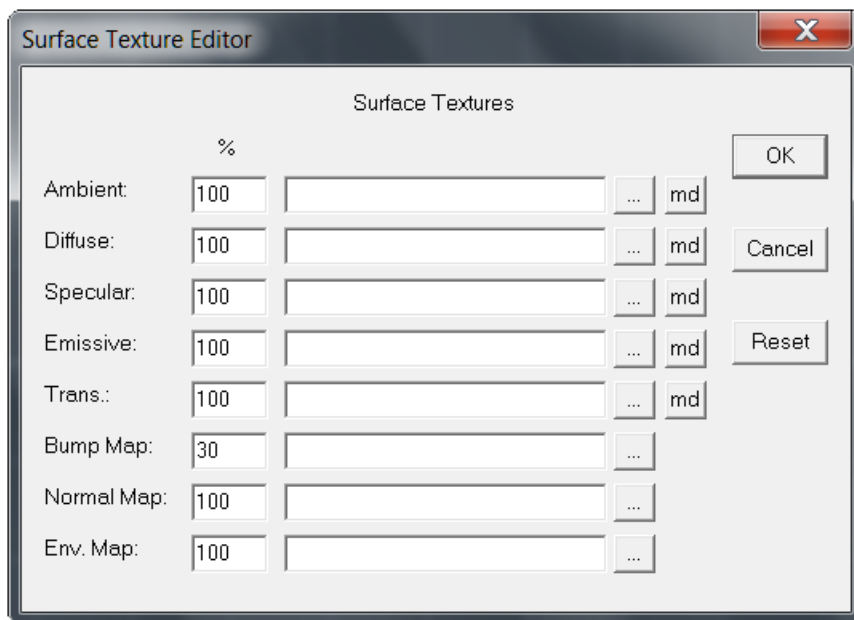
When you apply "materials" to your object they are applied smoothly and uniformly. However "texture images" use UV coordinates so when you apply it it may not appear right (ei: wrong size, multiple images, blurred color lines, etc.). It does not matter what size the actual texture image is, the UV coordinates specify a relative position in the image, what percent across or up and down, to apply to a face.

You can resize and adjust the texture image using the UV tool in the Point editor.

1. First select only the faces with the texture you want to rescale.
2. Then click on the UV button and
3. click Yes in the dialog to enable texture UV generation for the object.
4. Click drag in the middle of the big yellow circle on the screen with the MIDDLE mouse button to rescale the texture. Use the RIGHT mouse button to move it if it's off center.

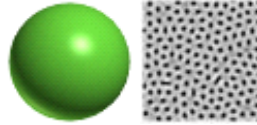
Advanced Textures

You can use textures for more than simple diffuse color maps. For example, you can map the transparency or shininess of a material to a texture. You can also use multiple textures on a material. The most basic textures can be selected directly from the material editor but you need the more advanced Surface Texture Editor for more exotic uses. You invoke this dialog using the  button in the material editor dialog:

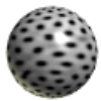


The example shows a normal diffuse texture named "spots" and a bump map texture named "noise". The net effect is a bump mapped, spotty material.

The following shows some of the surfaces that you can make with a simple green color and a single spotted texture using the different channels.



Above is the simple green material and the black and white spotted texture. By assigning this one texture to different channels of the green material you can create a variety of diverse materials.



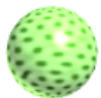
The **diffuse** channel is what you normally think of as the color of the material. If you specify a texture for this channel the material will take on its color, as shown on the upper material to the left, overriding the basic green color entirely.



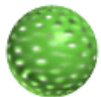
You can reduce the strength to something less than 100 per cent and the two will be blended accordingly. The lower material has a diffuse texture strength of 50, which means that the final diffuse color is a 50-50 blend of the basic green diffuse value and the image in the texture.



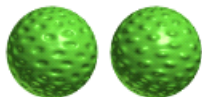
The **specular** channel controls the strength of the highlights. Using the spots texture here reduces the highlight where it is darker.



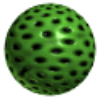
You can think of the **emissive** channel as glowing. Its strength is normally set to 0.0 so it has no effect. Here the emissive factor is set to 0.5 in the main material and the spots texture is assigned to the emissive channel.



If you use the texture in the **transparent** channel, the material will have holes in it where the texture is black and be opaque where it is white or very light



The **bump map** channel changes the surface normal to appear uneven, with lighter colored areas of the texture protruding and darker ones sinking. The surface of the object is still smooth, it's just a trick of the lighting that makes it appear uneven. The sphere on the left has a 30 percent weight assigned to the texture. The one of the right has a **negative** 30 percent weight which inverts the direction of the "bumps".



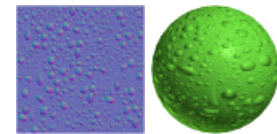
Here is an example of what you can do by assigning textures to multiple channels. In this case the same texture is assigned to the diffuse and the bump map channels, though you can just as easily use different textures. There is one more trick: the texture **blend mode** is set to darken which makes the diffuse channel the product of the texture and the basic green color. See the section on Texture Modes later in this chapter for more details.

Here are some more examples of using a single texture for both the diffuse color and bump map.
:



Normal Maps

A **normal map** is similar to a bump map except that the colors in the texture represent a rotation of the surface normal instead of the height of the surface. Here is an example normal map representing drops of water of a surface, and the same green material with it assigned to the normal map channel.



There are several ways to define normal maps. The most common uses tangent space normal. The red channel represents a rotation in the U or binormal direction relative to the surface, with 0.5 or 127 representing no rotation. Smaller values rotate the normal in the negative U direction, larger ones in the positive direction. The green channel does the same for the V or tangent direction.

Note: prior to versions 2.51 Blender used reverse U and V rotations, which make the surface appear indented where it should be raised.

Environment Maps

An **Environment Map** texture is a representation of the surrounding world. You use them to simulate shiny materials like chrome, and to show general reflections on an objects like the square shape of a window reflecting on a cue ball. Environment maps are not actual reflections of the rest of a scene. Instead they are intended to give the *illusion* that they are reflecting the





scene. For example, the image to the left uses an environment map of a view of mountains. The sphere appears to be made of chrome and reflects a view of the

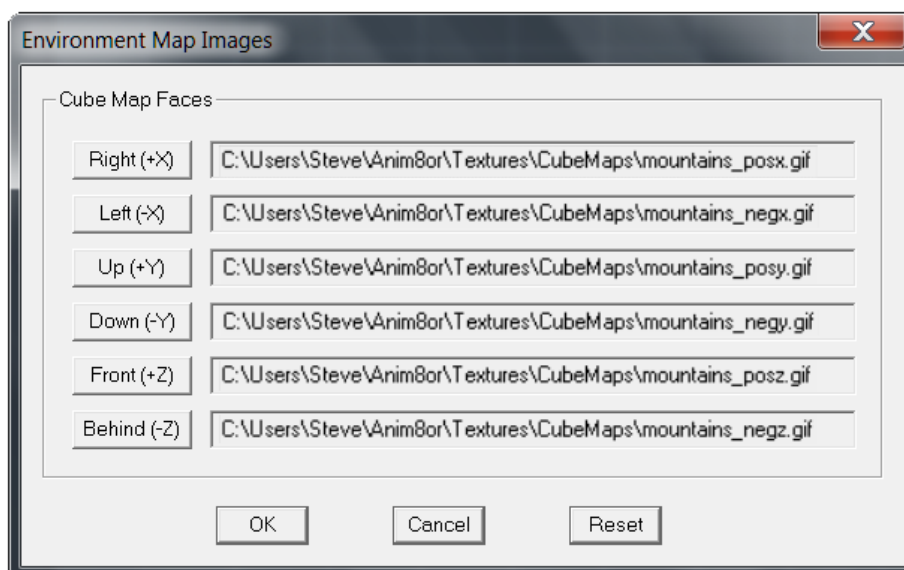
mountains and a chrome sphere would. It does not reflect either the checkerboard ground or the nearby 3D letters. These features are not present in the environment map so they aren't visible in the apparent "reflection".

The best way for you to create an environment map material is to use a **cube map** texture. These textures are composed of six different images that form the sides of a cube. Each one represents the view of the environment around your scene in a different direction. Think of it as if you are inside of a large cube. The six images used in the example above are shown below.



The directions are relative to the an object within the scene when the viewer looking into the **front** modeling view. So **behind** (-Z) means behind the object, in the -Z direction.

You add cube map textures by clicking on the  button on the Env. Map row of the Surface Textures dialog. This brings up the Texture Selector dialog in Environment Map mode. There you click on the usual Load Texture  button but instead of prompting you for a single file you are asked for six files:



There are a few things that you need to know about cube maps:

- The standard cube map format has the images flipped top-to-bottom. So if you find a set of cube map images that appear upside down don't worry. They will work just fine. However you can use a right side up set as well. Just check the "Invert Image(s)" box in the Texture Selector dialog and they will be flipped for you.
- Cube map images should be square. Rectangular images will not work properly, particularly in the working views.
- All six faces should be the same size. If any of the faces are a different size the texture won't draw properly.
- Cube maps require a lot of memory. Use smaller images if you can get away with it.
- As with all textures, you will get the best results if your images are a power of two in width and height. So make your maps 256 by 256, 128 by 128, etc. if you can.

Environment Map textures are always added to any the base material color. Thus you would normally set the color of a simple Chrome material to black if you are using an environment map. Of course you can experiment with other settings too, to see what they will make. Here are some other environment map textures:



You can also use a single image as an environment map. In this form the image is used as a latitude-longitude map of the environment. You can often paint a simple image to use for such a map.

The one at the right ...

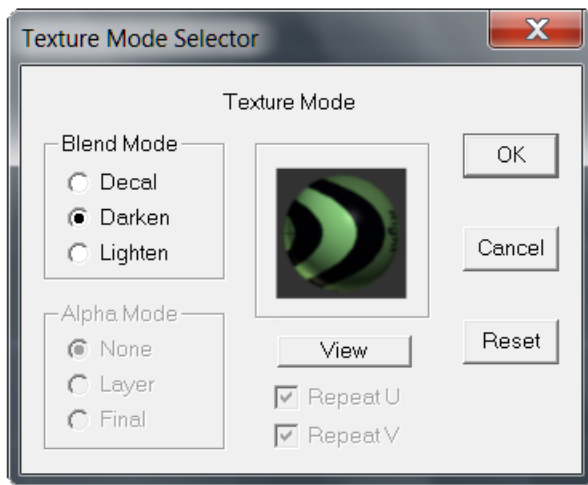


...makes a reasonable chrome reflection for an illustration.



Texture Mode

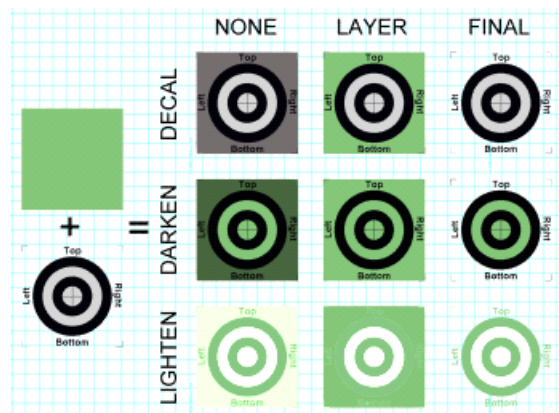
You can blend textures with the base color of a material in several ways. Textures with an alpha channel are even more versatile. You set the way textures behave in the texture mode dialog, which is reached with the **md** mode button in the material editor.



You use the **blend mode** to control how the color of the texture changes the base material color. **Decal** simply replaces the color with the texture. **Darken** multiplies the value of each color component which tends to darken the image. **Lighten** adds the two values which can only make the image brighter

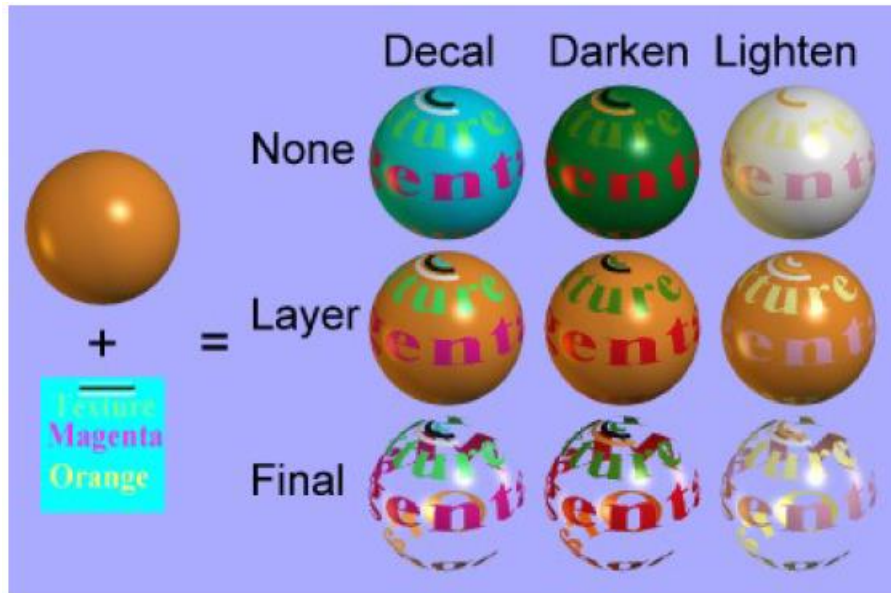
If the texture has an **alpha** channel (such as a transparent **.gif** has or an RGBA **.png** file) they you can use it in three different ways by setting the **alpha mode**. **None** simply ignores the alpha. **Layer** uses the alpha channel to blend between the texture's color and the material's color. **Final** uses the alpha channel as the material's final transparency value.

Below is an example of what can be done with one texture in the diffuse channel and a surface color of green. The background in the target texture, where the blue background grid is visible, has an alpha value of 0 so it is transparent.



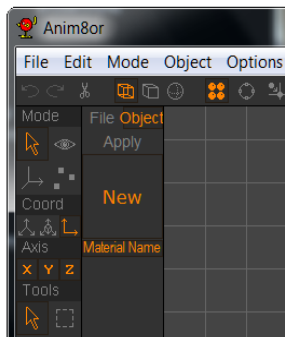
This is another example of what can be done with one texture and one surface color. The cyan

(light blue) color of the texture is the transparent color in a .gif file.



Some aspects of a texture mode are only meaningful for specific texture uses. For example bumpmaps don't use an alpha channel so the Alpha Mode set-tings don't have any effect on them.

Material Layers



Materials are bound by name. There are two layers, **file = global**, **object = local**. When an object's material is bound Anim8or first looks in the object material list for the name. If it finds it that is the material used. If not, then Anim8or looks in the file material list.

Here's my logic for designing materials this way:

1. There is a need for materials local to each object. Putting all of the materials in one giant pool quickly becomes unwieldy for large projects. So there are object-level materials.
2. There is a need for using the same material in multiple objects. Making changes to this material is difficult if you need to edit each one individually. So there are global-level materials.
3. If you import an object with a large number of materials and some of them have the same name as an existing material, how should it be handled? Anim8or's two layered scheme avoids requiring the user to rename a lot of the local materials whose names conflict with global material names.

Scripts

You can extend Anim8or's basic functionality and automate repeated tasks with **scripts** written in a programming language built into Anim8or. It is designed to make animation easier with features unique to Anim8or's operation.

There are several places that you can use scripts and they are all written in the same basic way. While they are all written in the same basic way, certain aspects are only allowed for specific kinds of scripts. For example you can't refer to a Figure in the Object Editor, and the time() function is not meaningful outside of a Scene.

Scripts may be used in the following places:

- Object text file export plug-ins,
- Parametric Shape plug-ins,
- General Object construction, editing and rendering in the Object editor,
- Controller expressions in a Scene.

There are other useful places for scripts that will be added in future releases.

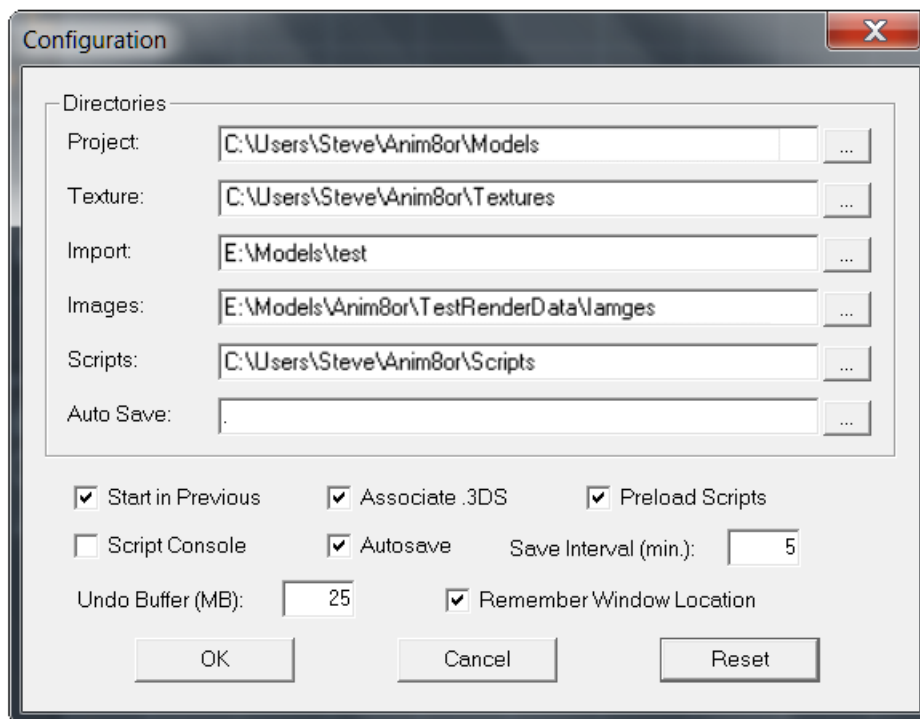
This chapter provides an overview of scripts and the basics of using them in Anim8or. A more detailed language definition is provided in the *Anim8or Scripting Language Manual*.

Creating a Script

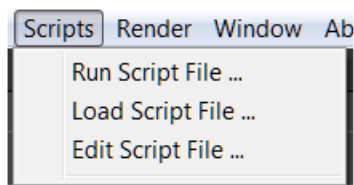
A script is a text file that has either a **.txt** or **.a8s** extension. You may use any text editor to create and modify them, or you may create them within Anim8or. Normally a script will have a .txt extension. If it has a .a8s extension and you place it in the "Scripts" directory set in the **File→Configure** dialog Anim8or will attempt to preload it when it begins execution. If the script is appropriately defined it will become a plug-in. Alternately, it can be added as a command in the appropriate Scripts menu for easy execution when you are modeling or animating.

The Script Directory

You may configure Anim8or to preload scripts from a directory called the scripts directory when it begins execution. This is where you put any plug-ins that you want to use and common modeling scripts. You set the directory in the **File→Configure** dialog and need to make sure that the **Preload Scripts** box is checked.



When preload is enabled Anim8or will read any files in this directory that have the extension **.a8s** and parse them as scripts. If a file has no errors it will be added to the Anim8or interface. Parametric shape plug-ins will go in the left hand toolbar. Object export scripts will be added as additional choices in the **Object**→**Export** dialog and Object command scripts will be added to the Scripts menu. All other files with the .a8s extension will be parsed for correctness but not saved in the computer's memory. You can run them with the **Scripts**→**Run-Script-File** command at any time.



Running, Loading, and Editing Scripts can be accessed from the Main Menu.

Installing a Script

All you need to do to install a script is to copy it into the scripts directory that you have chosen. Anim8or will load it when it starts running. It will also list the scripts read in the command window and any errors that it finds.

You may also store other scripts in this directory for convenient access but not have them preloaded by changing the file extension to .txt. You can edit and run them from within Anim8or with the **Scripts**→**Edit-Script-File** and **Scripts**→**Run-Script-File** commands. You can also load modeling scripts with the **Scripts**→**Load-Script-File** command. It's not possible to load additional plug-ins while Anim8or is running.

Plug-in Scripts

Plug-in scripts work exactly like commands that are built into Anim8or. The only difference is that they are written in the Anim8or scripting language and can be added or removed as you wish.

A plug-in script must be written to perform a specific task. Early in the script it will have a directive declaring what kind of plug-in it is and other related details. This is how Anim8or knows where to "plug it in". Additionally each kind has specific rules that it must follow or it would not work correctly.

Informational note: If you make a plug-in button for the menu bar you should remember that the maximum icon size is 26x26 - that's the size of icons in Anim8or (in pixels).

Parametric Plug-in Scripts

Parametric plug-in scripts allow you to add new basic modeling shapes to the Object editor. They are similar to the built-in spheres and cylinders in that you define their shape by a few numeric values. Anim8or executes a parametric mesh's script with updated values each time that it needs to update the geometry.

It's important to understand that when you save an .an8 project that uses a parametric shape plug-in you will need to have that plug-in installed when to properly view your model at a later time. You can load and edit a project with a missing shape plug-in but you won't be able to see or alter those shapes during that session. This is not the case for modeling or export plug-ins. Once you run of these they will make a change to your project or write an output file just like you used other similar commands.

Export Plug-in Scripts

Plug-in scripts that export objects appear in the **Object→Export** command's "Save as type:" drop-down menu along with the built-in exporters. They are indicated by the addition of ":plug-in" after the description.

Installing Plug-Ins

To install an Anim8or plugin script:

- Define the Scripts directory in the **File→Configuration** dialog,
- Check the "**Preload Scripts**" box, and
- Copy the script into the Scripts directory. Make sure it has the file extension **.a8s** or it won't install.

Running a Modeling Script

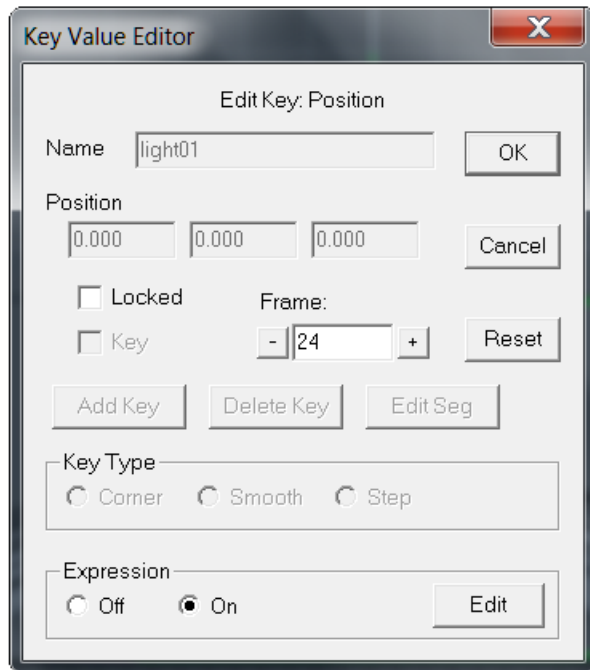
A modeling script is simply a script that is not a plug-in. They contain commands like you would use to build a model or make a material. When you run a modeling script it will change your project it some way and exit so that you can continue working. Normally this will happen very fast but it is possible to write a script that will take a long time to complete. Once you've become familiar with a particular script you will know what to expect from it.

You run modeling scripts from the Scripts menu. Those with .as8 extensions from the scripts directory will be listed in the menu. You can run other scripts from anywhere on your computer with the **Scripts→Run-Script-File** command.

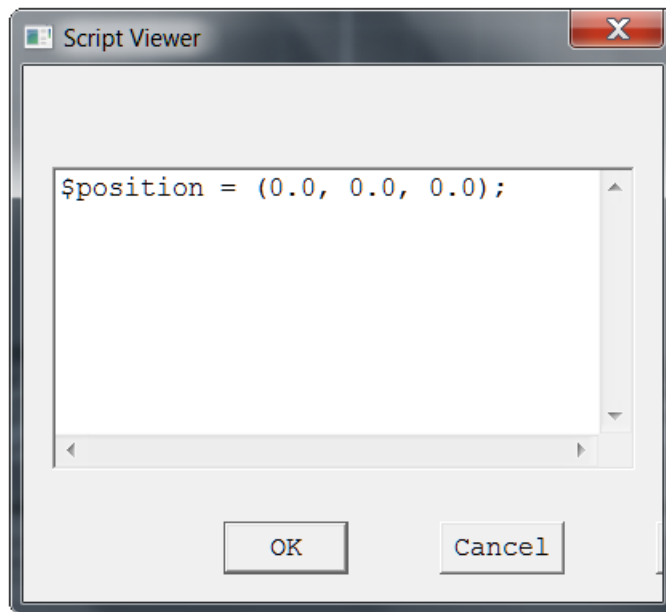
Controller Scripts

You can also use scripts instead of key-frame splines in controllers. To change a particular

controller to use them select the "On" button in the Expression area in the key frame dialog. (accessible via the **Scene→Element→→Key Value Editor** command). Anim8or will supply a very simple script to set the value of the controller, normally to 0.



You may edit the script with the Edit button.



Script Errors

If Anim8or detects an error while running a script it will print a message to the command window. If you are able to write scripts this may help you find and fix the problem. If you are running a plug-in then you should contact the author for a fix.

Writing Scripts

Script writing is simply writing a program in the Anim8or Scripting Language, ASL. You can

learn to write simple scripts fairly easily, particularly controller expressions. Complex scripts can require advanced programming skills. This manual does not explain how to write scripts. However if you know how to program in C or Java you should have no problem writing scripts.

A simple summary of the major aspects of Anim8or's scripting language is described in the following sections. The full syntax for scripts is described in the *Anim8or Scripting Language Manual*.

Data Types

There are quite a few predefined types in scripts. You will recognize some of them but many are specific to Anim8or. Some common types are:

- int - 32 bit signed integer
- float - single precision floating point number
- point3 - a vector of 3 floats
- object - an Object
- figure - a Figure
- material - a Material
- shape - a component of an Object such as a Mesh or Sphere

Variables

Scripts use typed variables. You must declare a variable and give it a type before you can use it. Built-in variables and functions use normal C rules for their names. That is they begin with a letter or an underscore "_" which can be followed by letters, underscores and digits. Examples are:

```
scale cos AddPoint face2 KEY_FRAME _exit
```

User variables begin with a \$ character followed by a C name:

```
$counter $NewIndex $face2 $_selected_index
```

Variable declarations are like those in C. Variables can be single values or arrays. Unlike C, you cannot initialize a variable in a definition however.

```
float $size, $width;  
int $i, $counter;  
shape $mySHape;  
int $index[10]; /* array initially of 10 ints. */
```

Arrays are much more flexible than those found in most languages. You can change the number of elements they hold in a running script, add and delete elements, and do other operations. There is more about arrays later on in this chapter and in the reference manual.

Expressions

ASL supports many of the operators found in C.

Function Calls

ASL has a number of built in member and non-member functions. These are described elsewhere in this document. User functions must be defined before they are called and recursion is not supported.

Unary Operators

- negation: int, float, point2, point3, quaternion
! not: int, float
~ bitwise not: int
++ -- prefix increment, decrement: int, float; note: returns l-value

Binary Operators

+ addition: int, float, point2, point3, quaternion
- subtraction: int, float, point2, point3, quaternion
+ concatenation: string
* multiplication: int, float, point2/3*float, float*point2/3, quaternion*float, float*quaternion
/ division: int, float
% mod: int
<< >> shift: int
< == <= comparisons: int, float, string
> != >= returns int with a value of 1 or 0
&& logical AND: int, float, returns int 1 if both operands are non-zero
|| logical OR: int, float, returns int 1 if either operand is non-zero

Statements

Scripts support several kinds of run time statements. Many are similar to C but not all. Statements can be nested to any depth. There is no goto.

The basic statement is an **assignment**. It is like C's except you cannot cascade multiple assignments in the same statement. The value of the expression is computed and saved in the variable at the left:

```
<variable> = <expression>;
```

Compound statements group several statements together into a single statement. When encountered each statement within a compound statement is executed in order. They are useful when you need to use more than one statement as part of a structured statement.

```
{  
<statement>  
<statement>  
...  
<statement>  
}
```

An **if statement** allows you to conditionally execute one or more commands. There are two forms one with an else and one without one. The expression is first evaluated. If it is not equal to zero then the <then-statement> is executed. If there is an <else-statement> it is skipped. If the value of the expression is zero then the <else-statement> is executed if one is present. The full statement is:

```
if ( <expression> )  
<then-statement>  
else  
<else-statement>
```

and the form without an <else-statement> is:

```
if ( <expression> )  
<then-statement>
```

A **while statement** repeatedly evaluates an expression and executes a statement as long as the value of the expression is non-zero:

```
while ( <expression> )  
<statement>
```

A **for statement** executes its sub-statement multiple times. For statements are not like those in C. In a script the increment and limit used with the control variable are computed prior to executing the sub-statement and saved. These values are used each time the body of the for statement has been executed to update the control variable and to test to see if the loop should be executed again. The syntax of a for statement is different from C to help remind you that it does not follow C's rules. There are two forms. The one without the <step-expr> uses a step value of 1:

```
for <variable> = <init-expr> to <limit-expr> step <step-expr> do  
<statement>  
for <variable> = <init-expr> to <limit-expr> do  
<statement>
```

The <variable> must be declared prior to the for statement and must be an int or a float. Initially the <init-expr> is evaluated and assigned to the <variable>. Then the <limit-expr> and <step-expr> are evaluated. If there is no <step-expr> a value of 1 is used. The value of <init-expr> is compared with that of <limit-expr>. If it is greater and the value of <step-expr> is greater than zero, or if it is less than and the value of <step-expr> is negative, or if the value of <step-expr> is zero then the execution of the for statement is complete. Otherwise the <statement> is executed, the saved value of <step-expr> is added to <variable> and they are tested in the same manner to see if the <statement> should be executed again.

User Functions

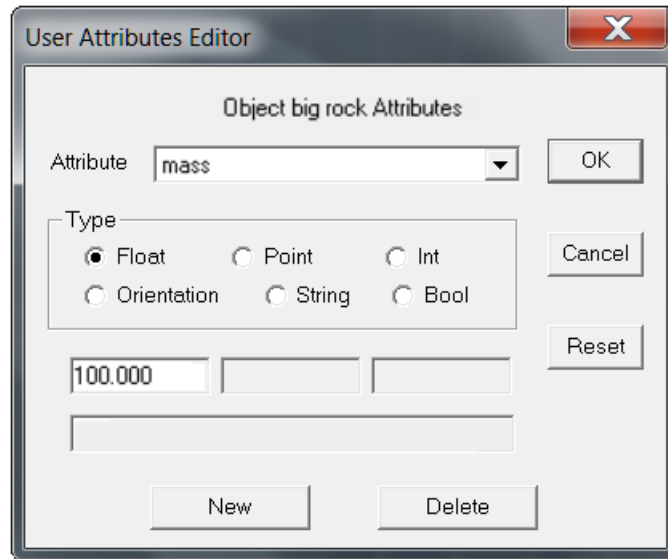
User defined functions are similar to those in C, with two main exceptions. Functions must return a value, and recursion is not supported. The syntax of a function is:

```
<type> <ident> ( <argument-list> )  
{  
<statements>  
return <expression>;  
}
```

You can have multiple return statements. You should have one on all exits from the function or the result will be undefined.

User Attributes

A **user attribute** is a custom property that you can add to objects, meshes, materials, etc. An attribute associates a name with an int, float, point3, quaternion or string value. For example you can define a float value called mass and set it to 1.5. They can be animated and useful as an auxiliary source of data for scripts. The dialog for defining attributes is shown below:



The attribute "mass" is a float with the value of 100. Anim8or doesn't understand these values but you can query and change their value in controller scripts to control how your characters behave with the `GetAttributeFloat()` and similar functions. See the *Anim8or Scripting Language Manual* for details.

Forum Notes:

Sample Script

webpage_ <http://www.anim8or.com/smf/index.php/topic,4822.0.html>

ASL Functions to make glass

Steve

Administrator

Hero Member

March 05, 2014, 11:15:27 pm

Here is an example that builds a glass material using attributes:

```
material $MakeGlass(void)
{
    object $Object;
    attribute $Attribute;
    material $glass;

    $Object = project.curObject;
    $glass = $Object.NewMaterial("glass");

    $glass.diffuse = (0.96, 1.0, 0.98);
    $glass.Ks = 0.7;
    $glass.alpha = 0.1;
    $glass.roughness = 200;
    $Attribute = $glass.NewAttribute("class");
    $Attribute.SetStringValue("dielectric");
    $Attribute = $glass.NewAttribute("IOR");
    $Attribute.SetFloatValue(1.33);

    return $glass;
}
```


ART Ray Tracer

Ray Tracing is a global rendering technique for making relatively realistic images from 3D models. It traces the path that a light ray might take through the scene to discover what objects, lights, etc. affect the final color of a pixel. It is called a global technique because any part of the scene may influence to final value of any pixel via reflected rays of light. Ray tracing borrows from optics to model reflections, refractions and more.

Anim8or has an integrated ray tracer. You can use it to render movies and still images just like with the original scanline renderer and OpenGL shader renderer. All you need to do is select it from the **Render**→**Renderer** command dialog.

Anim8or's ray tracer can render all of its normal materials, of course. But it also can show reflections off of shiny objects and diffraction through transparent ones. The next sections will explain how to add these effects to your images.

Art Materials

You enable the additional capabilities of ART materials by adding certain **attributes** to normal materials. Attributes allow you to associate arbitrary values with a material. The ART renderer checks them when initializing a scene and chooses alternate rendering code when the right ones are present. You add attributes to a material by clicking on the "Attributes" button  in the basic Material Editor. This section lists the main categories of ART materials.

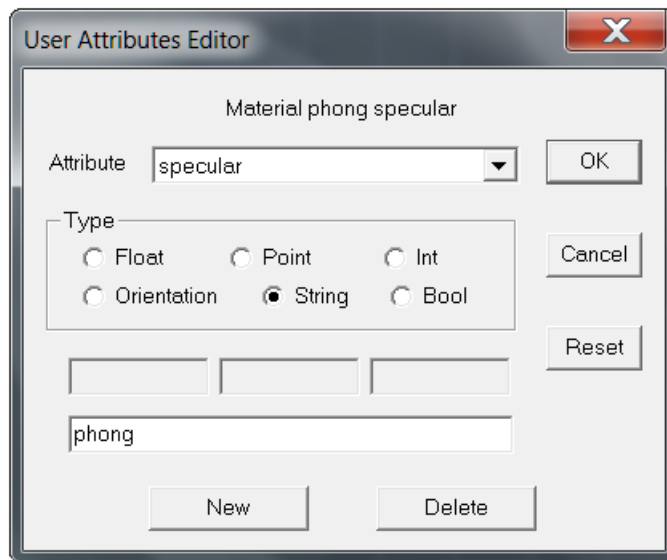
The attribute class is the starting point for accessing ART materials. class is a string attribute which defaults to normal Anim8or materials when not present. *Note that case is important in attribute names.* Class and CLASS are names for different attributes and do not influence ART rendering. In addition to the normal parameters of Anim8or materials, individual classes may use additional attributes for more flexibility.

The main classes are anim8or (the default), glossyreflector, transparent and dielectric. Each is described in a following section.

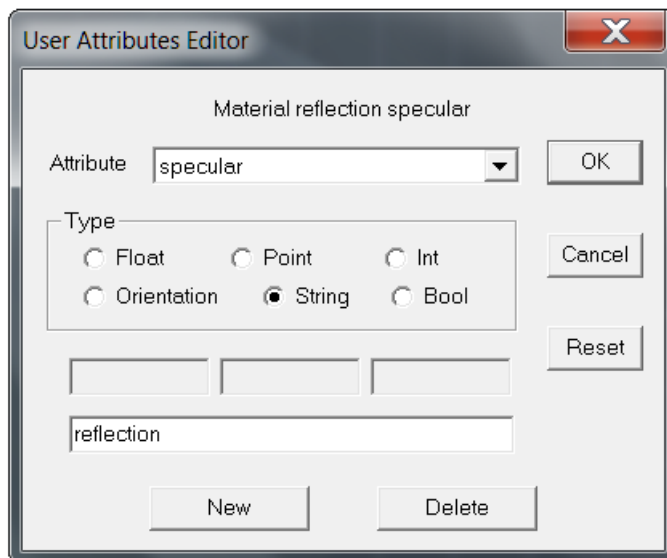
Class anim8or

Class anim8or has an additional string attribute, specular. It determines the look of the specular reflection. There are 3 recognized values:

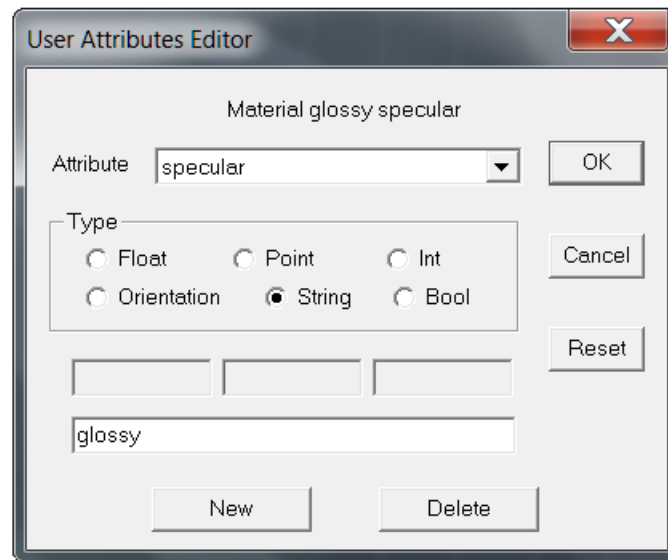
phong - the default value, phong is simple normal Anim8or phong shading for highlights.



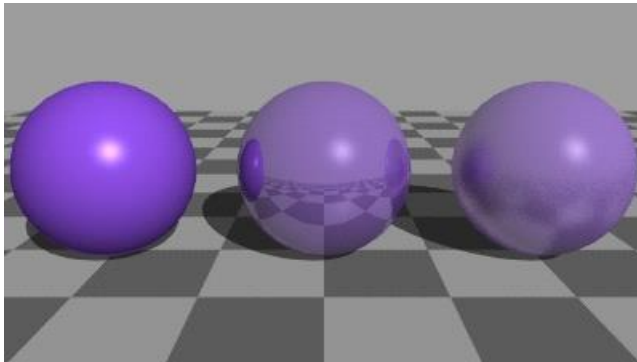
reflection - a mirror like perfect reflection is added to a material. Rays are reflected off of a surface to determine what is visible in that direction. The result is multiplied by the materials specular weight, K_s , the material's normal color - including the original phong highlight - is weighted by $(1 - K_s)$ and the two values are added for the final color. Thus a value of $K_s = 1.0$ would be a perfect mirror with no base material properties visible, and $K_s = 0.1$ would be a normal material look with a slight reflection of the scene.



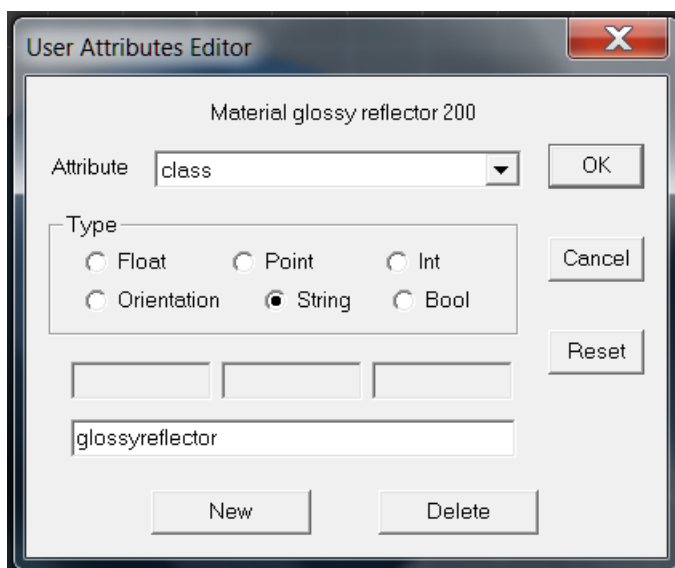
- **glossy** - an imperfect reflection is added to the material. This is like a blurred mirror or defocused camera image. The further from a surface, the less distinct the reflection. The overall amount of blur is determined by the Phong Roughness. The default value of 32 is very rough. Higher values like 1000 make a more perfect reflection while 100000 is almost the same as a perfect reflection. Glossy images need anti aliasing turned on or they will look very speckled. Extremely glossy images will need more than the default AA samples per pixel value of 16. The image below uses the default, 32, as the default samples per pixel of 16 and is rather grainy.



Below is an example of the three settings for the specular class. From the left they are phong, reflection and glossy. Ks is 0.5 and Roughness is the default of 32.



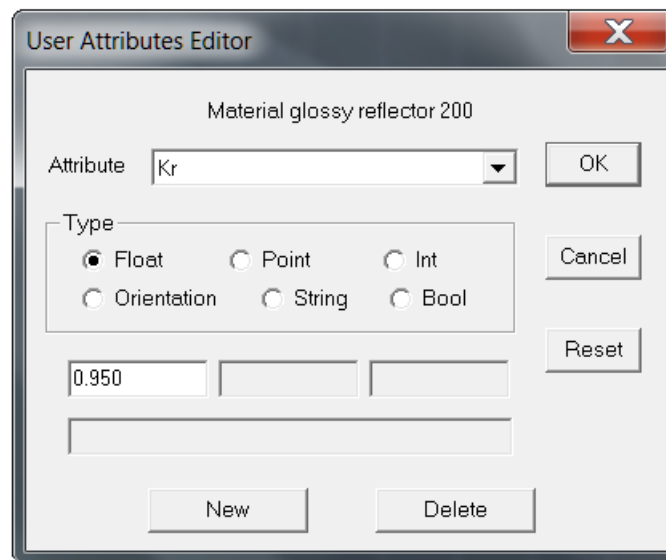
Class glossyreflector



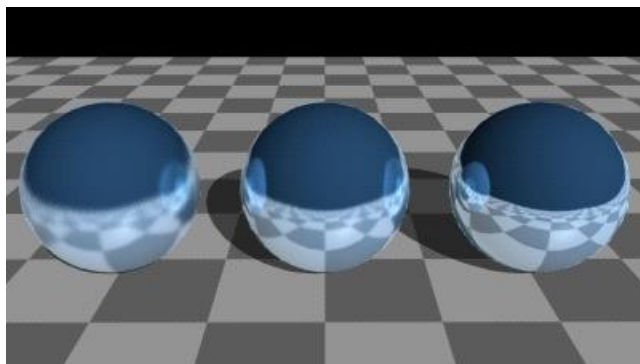
This glossyreflector class is a simpler class that is similar to the anim8or class with the specular attribute of glossy. While the anim8or class is a little less accurate from a physical sense, glossyreflector attempts to follow the rules of physics a little more closely.

The glossyreflector class uses the Phong roughness value for the degree of glossiness. glossyreflector has one additional attribute:

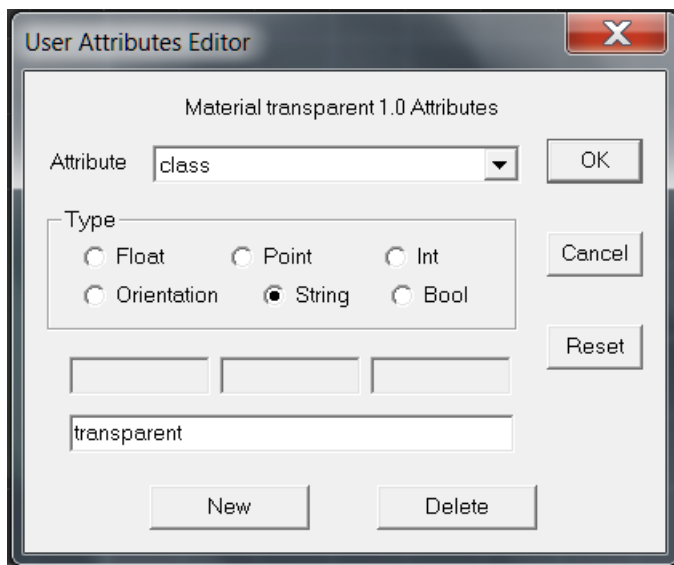
- **Kr** - constant of reflection. This is similar to the specular constant in normal Anim8or material's, Ks, but controls what percent of the light is reflected. It is set to the value of Ks if there is no explicit Kr.



Below are three glossyreflector spheres with roughness of 200, 1000 and 100000. You can still see quite a bit of graininess in the left sphere even though this image used 64 samples per pixel. Glossy surfaces require many more samples for high quality images. Also notice that there is no specular highlight from the infinite light source. Infinite lights don't represent real, physical entities. An area light would show a specular highlight however.

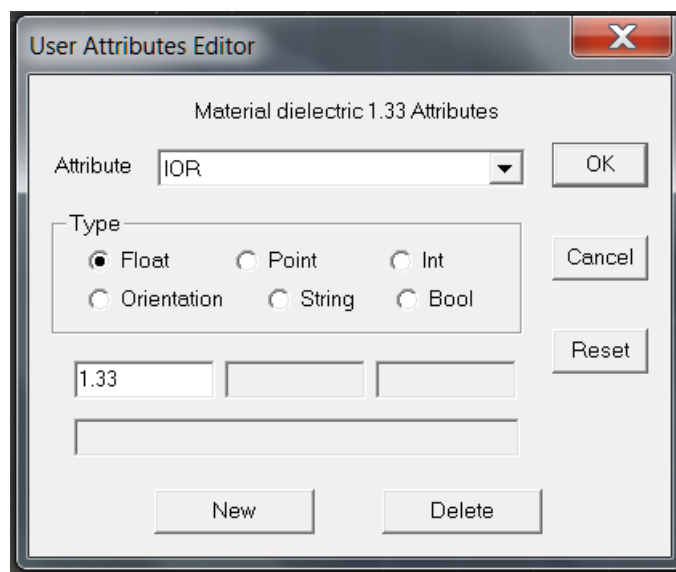


Class transparent



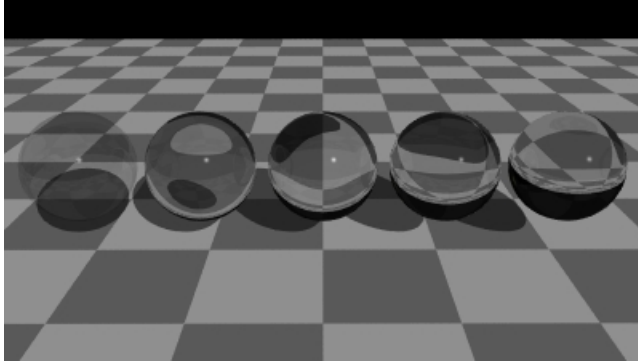
This transparent class is a simple model for diffraction of light through transparent materials. There is also a slight reflection off the surface. The amount of transparency is set with the normal transparency value for a material. It has one additional attribute:

- **IOR** - the index of refraction. This represents how much light is bent as it enters or leaves a transparent material. Useful values include 1.0 (air - no difference than outside of a material), 1.33 (glass), 1.49 (Plexiglas) and 2.42 (diamond).

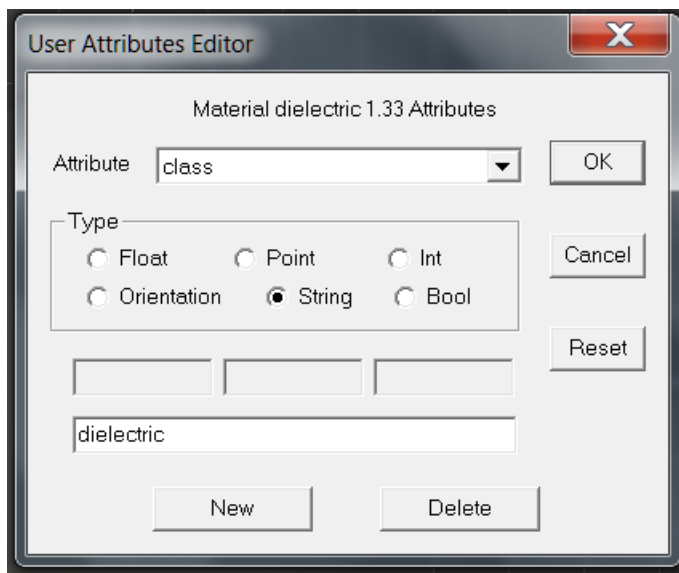


Rendering transparent materials can take a very long time. Each primary ray (from the camera) can trigger tens or even hundreds of rays since each interaction generates up to two child rays, the reflected ray and the transparent ray.

The image below show 5 spheres with and IOR of 1.0, 1.1, 1.2, 1.3 and 1.5, and a transparency of 0.1:



Class dielectric

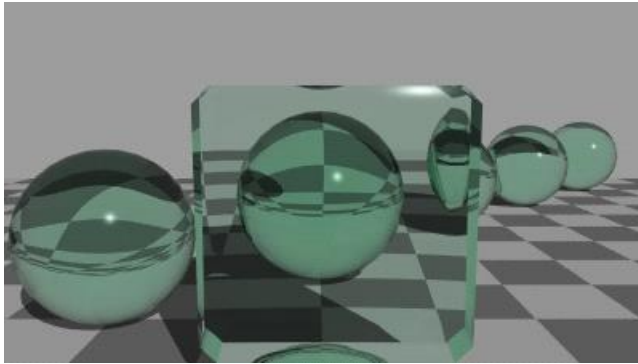


The dielectric class is a complex model that is closer to how light interacts with real transparent materials. It uses Fresnel equations to determine reflections and transmission parameters at each ray-surface interaction. As light travels through a dielectric material it is attenuated and gradually takes on the material's color. Thin sections appear almost colorless but thick sections can be quite dark. Glass is a dielectric which explains why a solid glass door looks clear from the front but looking into the edge can be dark green or gray. The dielectric class has two additional attributes:

- **IOR** - the index of refraction. As with the transparent class, this controls how much light bends as it enters or leaves the material.
- **UnitDistance** - the distance a ray must travel through the material to be filtered by the material's diffuse color. The ray's color is multiplied by the diffuse color raised to the power of (the distance traveled/UnitDistance). So if the diffuse color is (0.5, 0.5, 0.5) and UnitDistance is 10, then a ray that travels 20 units will be scaled by $(0.5, 0.5, 0.5)^2$ or

(0.25, 0.25, 0.25). The default value of UnitDistance is 10.0.

The image below shows several spheres and a rectangular prism that are dielectrics with an IOR of 1.3 approximating glass:

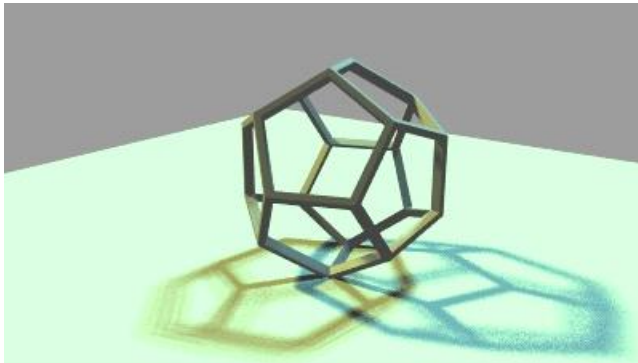


Light Attributes

You can modify the look of area lights by changing the way that they are sampled with the following attribute on the light in a scene:

- **sampler** - is a string attribute with three recognized values: multijittered (the default), jittered, and regular. The jittered modes produce a more random or grainy effect when under sampled while the regular sampler looks more like multiple shadows.

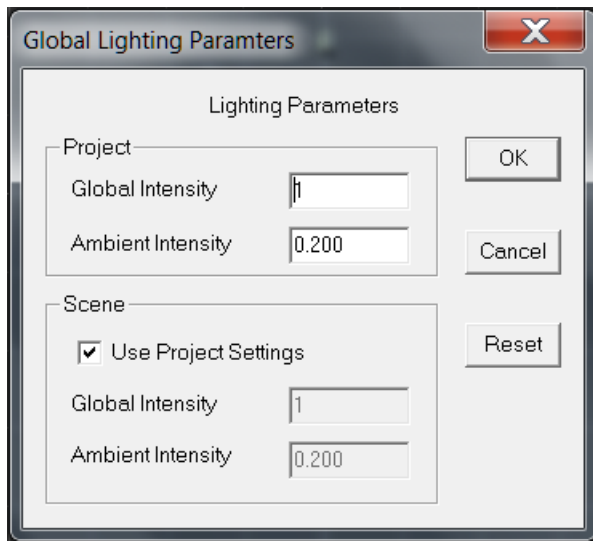
The image below has two area lights, the blue one (the yellow shadow) uses a regular sampler while the white light (which makes a blue shadow) uses the default multijittered sampler.



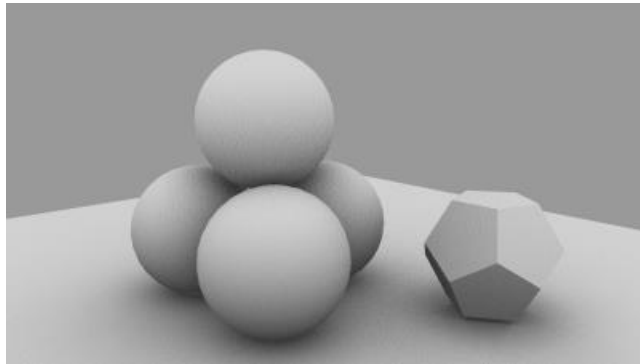
Ambient Occlusion

Ambient occlusion approximates light reflected from surfaces other than light objects. It adds a realistic aspect to soft shadows in corners and where objects meet.

You enable ambient occlusion in Anim8or by setting the Scene int (integer) attribute AmbientOccluder to 1. Together with the Global Lighting Parameters of **Global Intensity** and **Ambient Intensity** (accessible via the **Scene→Properties→Global Lighting** command) ...



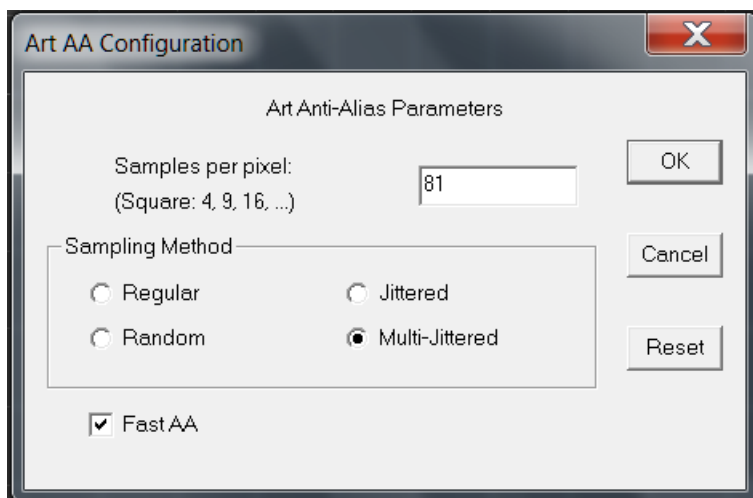
... you can render realistic corner shadows such as shown below:



There are no lights in this scene. Only the color of the background was used as a source of light. The settings were: Global Intensity of 0.0 (equivalent to no lights used), Ambient Intensity of 3.14 (required for correct lighting using background only), the int Scene attribute AmbientOccluder with a value of 1, and 256 AA samples/pixel.

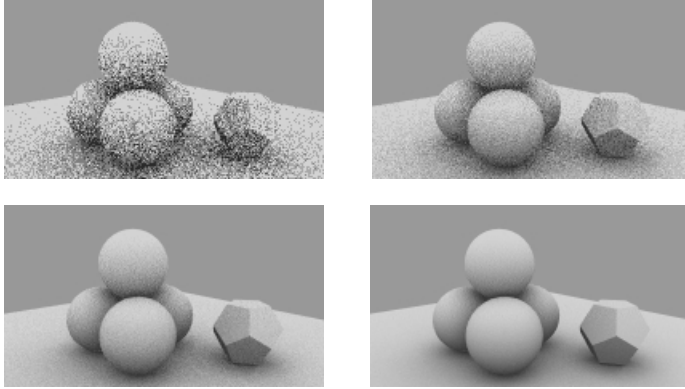
* Anti-aliasing **must** be enabled for Ambient Occlusion.

Ray Tracing and Anti-Aliasing



(accessible via **Render**→**AA Setup...**)

Ray tracing uses statistical sampling techniques to render global lighting, reflections, transparency, diffraction effects. High quality images require multiple sample rays per pixel. This, of course, means longer rendering times. There is no single minimum number for a "good" image. You'll have to experiment to find what works for your scenes however they must be a 'squared' number. Ambient occlusion using only the background color, as in the image above, can be particularly compute-intensive because light needs to be sampled from all directions. Here are three images of the same scene rendered with 4, 16, 64 and 256 samples:



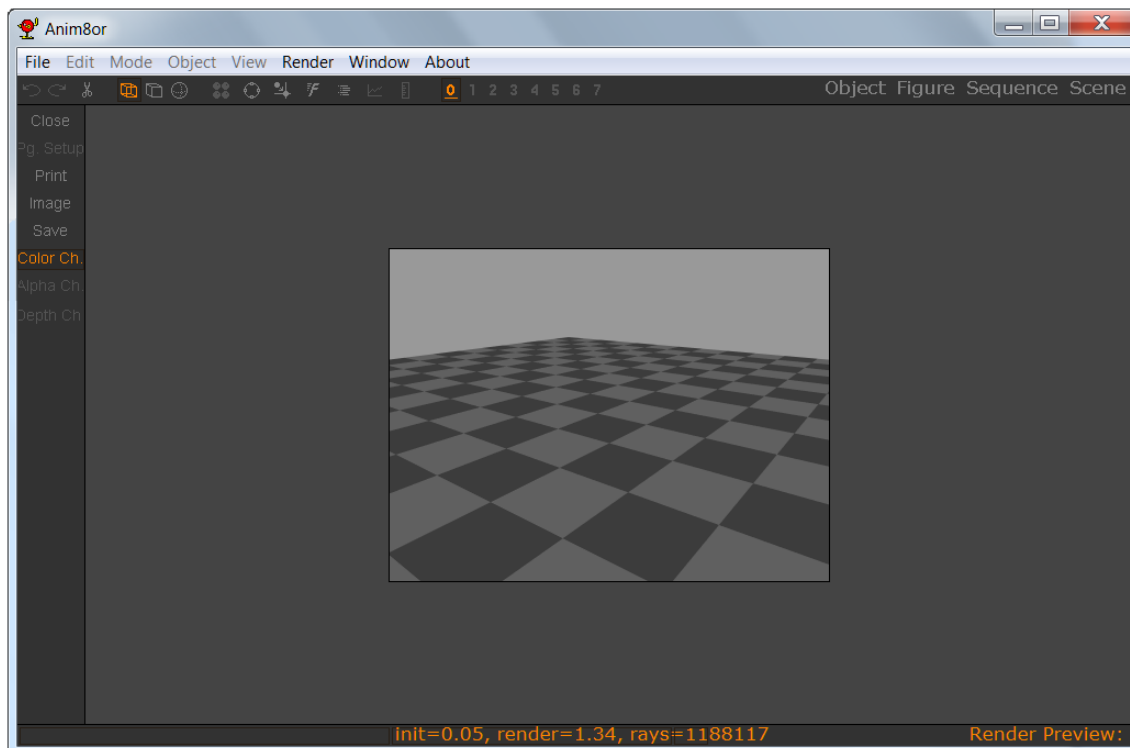
Rendering

Rendering Images and AVIs

Irregardless of whether you are rendering an image or an AVI, after you finish with your parameter settings you will eventually end up at the **Render Screen**.

In the **top** menu and toolbar you will find that you have limited choices related only to rendering. You will find the *File, Render, Window, and About* option buttons are available.

Along the **bottom** you will see the progression of your render and finally at the end, information regarding your render. Initially Anim8or may seem to 'hang' on you or appear to crash your computer however at the beginning it is doing its initial work to get ready for the rendering. The time varies according to your image or AVI, your version of Anim8or, and to your hardware. (example: Anim8or 0.95c would appear to hang for 5 minutes for each and every hour of rendertime. An hour (60 minutes) hangtime meant roughly a 12 hour rendertime; Scanline.).



Render Screen

On the **left** side of the render screen you will find your various command buttons that become available once the rendering is complete. Listed below is their brief descriptions:

- **Close** - Closes the render screen and takes you back to scene mode (or other mode that you were in)
- **Pg Setup** -

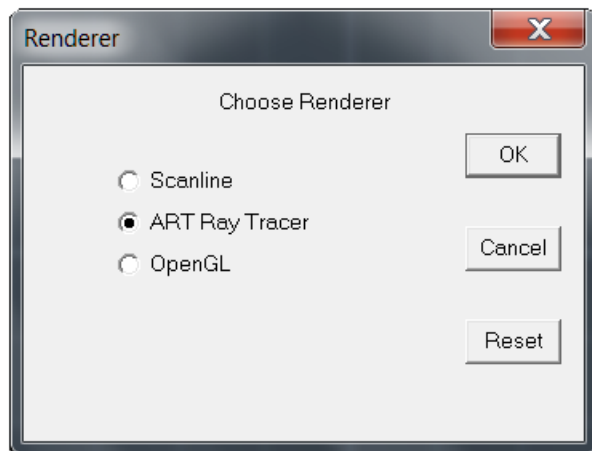
- **Print** - Opens up your computers printer choice dialogue box where you can choose your printer.
- **Image** - This button will take you back to the Frame Spec Editor so you can change or tweak its' settings and try again.
- **Save** - Opens up the Image File Format Editor so you can save your image.
- **Color Channel** - Shows you your color image in the Render Screen (this is the default). Anim8or uses the *RGB color space*.
- **Alpha Channel** - Shows you your alpha map image in the Render Screen. The *Alpha channel* is described in detail below.
- **Depth Channel** - Shows you your depth map image in the Render Screen. The *Depth channel* is described in detail below.

Not shown is that anytime during the rendering whether it is an image or an AVI it can be cancelled by pressing the **ESC** key on the keyboard. Be patient as it may take a while.

Renderer

One of the first options you will have to choose will be your renderer. ART Ray Tracer is the default so this is what you will be using if you forget. Don't be afraid to try the others to see the difference.

You choose the renderer in the Menu Bar **Render**→**Renderer** dialog box.



Render Options

Listed here are your options:

- **Scanline**
Scanline is the line by line method of rendering. The part of the scene that intersects with the line being scanned is determined as to what can be seen to be rendered and then, when finished this information is no longer used. This saves time and memory. This method is used quite often for real-time rendering such as in interactive games. This is NOT the same as z-buffering! Since scanline rendering does not use ray tracing the appearance of transparent objects and material details may not be photorealistic. Even so, many scanline renderings are quite breathtaking and quite useful.

- **ART Ray Tracer**

Anim8or also has a built-in ray tracer renderer called ART for Anim8or Ray Tracer. This is described in detail in **Chapter 11: Art Ray Tracer**.

- **Alternate Renderers**

If your graphics card supports advanced shaders you may choose to use an alternate OpenGL renderer that uses the graphics accelerator in your computer to make images and movies. This renderer is much faster than the default Scanline renderer but it does not support all of Anim8or's options. Most notably, shadows are not supported.

If the "OpenGL" entry is not present Anim8or cannot render using your computer's graphics hardware.

Rendering Images

Rendering your Image

Whether you just want a preview of how your image is coming along or you have finally done all that work and now are ready for your final render you can set everything in motion by going to the Menu bar under **Render**→**Render Image**. The frame Spec Editor **Frame Parameters** dialogue box will open up to show you the choices available to you. If you would like to find out a little more information that will help you with your choices regarding the three standard image formats in Anim8or and the common image resolutions that are used you can go to **Appendix B: Standards and Aspect Ratios**.

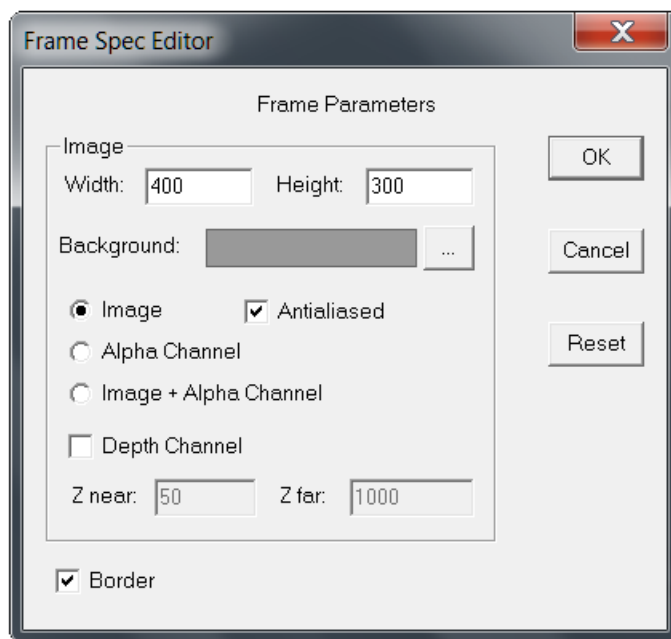


Image Frame Spec Editor

Frame Parameters

Image Resolution

Image Resolution is the actual size of your image not the aspect ratio. If you only want a preview to check something then you can use the quick defaults in Anim8or. They are 400 x 300 (4:3) and 400 x 225 (16:9). Otherwise just put in the numbers of your choice. The numbers are in pixels.

Antialiasing

Aliasing is an effect that under some circumstances causes artifacts, distortion and moire effects in your image. This can also be present in your AVI causing motion optical illusions. To counter this make sure your antialiasing is checked (default) but you do have the option to turn this off if you are attempting to obtain something specific.

Background


If you are doing a preview of a model or have not made choices in the Environment Editor as of yet this is where you can set an appropriate solid color for your background. The default is gray but you can change this using the  button on the right.

Image Channel Choices Here you choose which channels that you want rendered. Your Image will be by default because it is the color channel. The other channels are described below in the next section **Channels**.

Znear/Zfar

If you are unsure of this you can leave it however it is described in detail in **Chapter 7: Scene Editor**.

Border

The choice of whether you want a border around your final image or not.

One thing you should always remember: Anim8or renders from the *highlighted screen* whether it's the Camera viewport or another. This is true for both Image and AVI renders so double-checking which viewport you are in could save you a bit of time and trouble.

Saving your Image

If you decide to keep your rendering after it has completed then press the **Save** button on the left sidebar. This will open up the **Image File Format Editor** to set your Image Parameters.

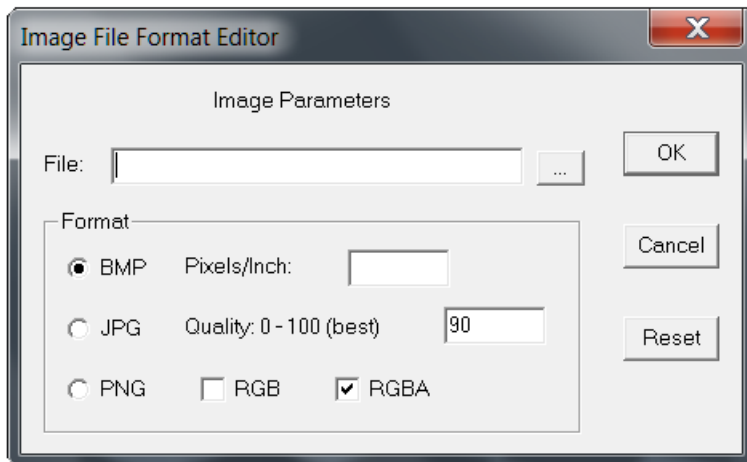



Image File Format Editor

File

Here you can name your file. Don't worry about the extension for the Image Standards Format because Anim8or will add them for you. These images will be saved in the Image Folder that you have listed in your Configuration dialog (**File**→**Configure** or see **Chapter 2: Basics**) If you have not configured this or you would like to use a different folder (such as a special project

folder) click on the Browse  button on the right and find the folder you wish to use. After you have saved it the location and filename will stay in the textbox in case you wish to re-render. You can also rename it by highlighting the text and changing it.

Warning: You will not be asked if you want to overwrite another file if it has the same name such as if you are doing a second render that you also want to keep.

Remember that if you are also saving its' Alpha or its' Depth channels they will not be saved automatically. To save these click on the appropriate channel button so that the image appears in the viewing area of the Render Screen. Click on save again and this time you will find that your alpha or depth image is already in the filename textbox and it includes an addition to the name showing which channel it is. Just press okay and it will be saved along with your Color Channel image.

Anim8ors Image Formats

Here you choose one of the three Image file formats available in Anim8or. Once again, there is more information in **Appendix B: Standards and Aspect Ratios**.

- **BMP (Bitmap)**
Anim8ors' default : 72 pixels/inch
Bitmap is the **uncompressed** standard for raster graphic images
- **JPG (Joint Photographic Experts Group)**
Anim8ors' default : Quality 90 %
JPEG is a **lossy** standard that is widely used for photographs and raster graphic images.
- **.PNG (Portable Network Graphics)**
Anim8ors' default : RGBA
PING is currently the most commonly used **lossless** standard of raster images. It includes both 24-bit RGB and 32-bit RGBA (*RedGreenBlueAlpha*).

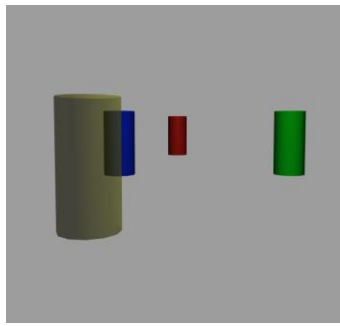
Channels

The Alpha Channel

When you render an image, you can isolate the elements in your scene from the background. You can save both the images of your elements and a mask showing where they are. This mask is called an **alpha channel**. It's really nothing more than a black-and-white image in the shape of your elements. It has a value of 0 where your elements are drawn, and 1 where they aren't. Transparent parts and edges in anti-aliased images will have a value in-between representing how much of the pixel your elements "own". You can use the alpha channel to combine images made in Anim8or with other images in a paint-program, as you will see below.

You make an alpha channel by selecting the "**Alpha Channel**" button in the preview dialog.

You can also make two very useful images with one rendering by selecting the "Image + Alpha Channel" button. One is an ordinary alpha channel and the other is a special element image that has been **pre-multiplied** by the alpha channel. It shows your elements normally but is black where your elements are not. Partially transparent parts of your elements are dimmed to the degree that they are transparent. You can then use these two images for a variety of special effects, such as image compositing which is described next.



Image



Alpha Channel

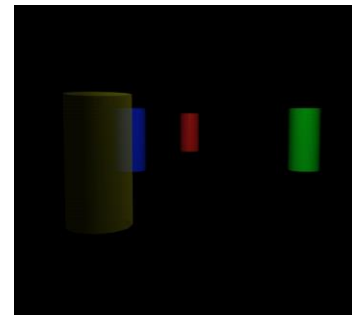
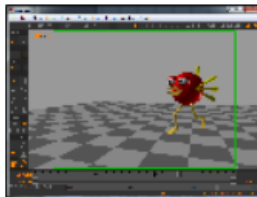


Image + Alpha Channel

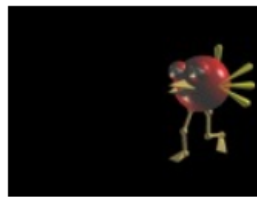
Image Compositing

With **image compositing** you can insert a computer generated model into an ordinary photograph or movie, or computer generated ones. You first render an "Image + Alpha Channel" image and then use a paint program such as Adobe Photoshop to add Anim8or's output to the photo. The steps in this process are shown below. It's best to enable anti-aliasing to prevent annoying jaggies in the result. If you would like to use your alpha mask in another way such as using it for another image you can simply **invert** the alpha mask in your paint program so the white areas become black (for transparency) and the black areas become white (for opacity).

Step 1: Render an "Image + Alpha Channel" of your model in Anim8or. The result is two images as shown here:



Camera view in Anim8or



Pre-multiplied by Alpha



Alpha Channel

Step 2: In a paint program, multiply the photographic image by the alpha channel image. This will mask out a hole in the photo for your model:



Photograph



Alpha Channel



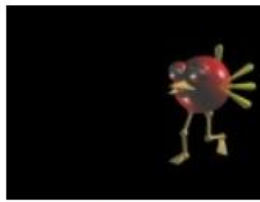
Masked Photograph

Step 3: Still in the paint program, add the Anim8or image to previous result:



Masked Photograph

+



Pre-multiplied by Alpha

&



Final Composite

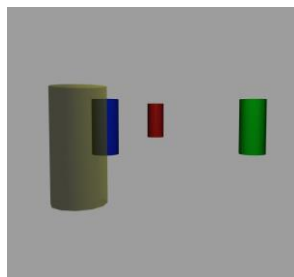
It's important to try and match the lighting on your model to that in the photo. Otherwise the computer generated part won't blend in very well. In this example the bobcat is lit by the sun from behind and to the right but the bird has some front lighting as well which makes it look artificial. (You would think it was a real bird otherwise, right?) Along with your lighting placement to help you match your model and your photo you also have to consider the color of the lighting you should use (or as close an approximation as possible). This is usually done by finding the area in your photo that could be, under normal lighting circumstances called "white". By determining the difference in this "white" area and its actual color in the photo you then have a base color for your lighting. (HINT: put your photo in a photo editor that has a color picker and find the value of the "white" area in your photo. That will be your light color.)

The Depth Channel

There are times when you may be needing a **depth map** or a **depth channel** for the project that you are working on. The depth map or channel is a specific image that holds the information about the distance from the viewpoint (example: the camera) to the various objects in your image. It is quite often referred to as the **z-depth map** . While your scene and/or your objects are oriented along the world or the objects **x-y-z** axis the depth map is created using the z axis of the viewpoint being used, whether your camera or a focal plane (for depth-of-field images). In other words your depth map is created while looking straight out from your (or the focal planes) viewpoint. You may notice what appear to be some discrepancies in the image. If you remember that the depth map is based on being **perpendicular to the focal plane (90 degrees)** what seem to be the distance to an object or even to the objects at the edge of your image is not the same because of the angle that they are being viewed at. This creates a difference in the distance related to the angle of the z axis of the viewpoint and so is shown in the depth channel as being a different distance. For most instances this difference is usually not noticeable to any degree.

You will notice that when creating a depth map using your viewpoint (camera) the closer that your objects are to the viewpoint the darker they appear. So therefore black is close and white is far away with the varying shades between being relative to those two. If you were to create a depth channel related to a focal plane then the darker colors would be closest to the focal plane and the lighter colors farther out from the plane. In this instance, where the viewpoint was not be considered, as the focal plane would be the point of origin for the depth map.

You make a depth map by selecting the "**Depth Channel**" button in the preview dialog.



Image



Depth Channel

What can you do with it now that you have it?

Anim8or creates a single pass depth channel. That means first come - first see ! The depth map created therefore only records the objects that are visible. It does not take into account properties that multiple pass depth channels do such as if there is an object hidden behind another and various information required by game-makers and some 3D programs to assist in z-culling. That is another subject that does not need to be discussed here. However, when you have a depth map you can use it in your projects to help with the visibility properties of water, smoke, fog and aid you in creating atmospheric objects such as clouds. It can also be used for creating height maps, stereogram images, and can be useful in shadow mapping and assist in creating organic material properties such as sub-surface scattering.

Be imaginative ... and have fun !

Rendering .AVI Movies

Now, after all that work you want to have your creation come to life. That invention that's absolutely amazing, the old ship sailing the seas, bizarre monsters from outer space, or simply a cute *Robin* wait for you to hit that render button! It is time for your animation, your video, or even ... **YOUR AVI MOVIE** to begin a life of its' own.

AVI format

.AVI (Audio Video Interleave)

Originally developed and released by Microsoft in the early 1990s the AVI format has both the audio and video data included to be played synchronously during playback. Allowing for the use of various codecs (software for the type of compression/decompression used: enCOding and DECOding) for the images and sound has produced a format that is one of the mainstays of the industry. It must be remembered that it even uses Uncompressed images and sound.

"WAIT JUST ONE DARN MINUTE" You say, "ANIM8OR HAS NO SOUND"
"No sound is STILL sound" I say, " It's called 'Dead-Space'" ! :)

Anim8or itself only produces the video portion of the AVI and leaves the sound as deadspace. Incorporating sound and sound effects with Anim8or's video can be done in a separate movie editor. Note that AVIs do not do well with sound files having variable bit rates. (HINT: While most home 'editors' for computer, television, or the Internet do not take this into account, professionals and many experienced home editors realize there is a time lag between the image and the sound reaching their audience. Depending on the 'target distance' this can mean setting the sound to start anywhere from 0 to 56 frames in advance of the image frame in order to synchronize them together at the 'target distance'. Know your target distance - know your time lag!)

Rendering and Saving your AVI

So, you've double-checked everything, made sure the right viewport is highlighted, and your drinks and chips are ready for the long wait. It is finally time to render that creation!

Scene Parameters

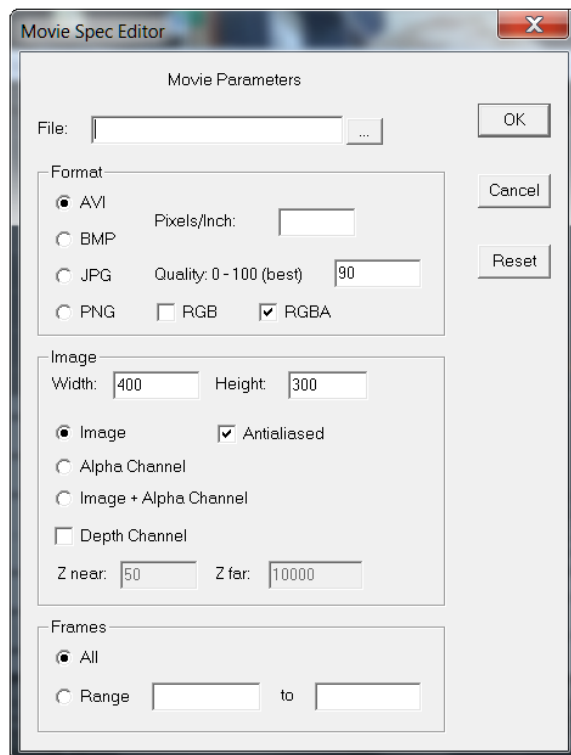
If you are rendering your final version you probably already have made your Scene Parameter

choices. If not then you can do that now. These can be found by going to **Scene→ Properties →Movie Image → Scene Parameters Editor**. See **Chapter 7: Scene Editor** which describes the Scene Parameters Editors' choices.

Movie Parameters

You can disregard this if you are merely rendering a test AVI however if you have not set up your movie properties such as the format and Aspect Ratio you still can make your choices. **Scene→ Properties →Movie Image → Scene Parameters Editor→Movie Image Properties** . These settings are transferred over to the renderer when it is the actual time for rendering. You may also want to have a look at **Appendix B: Standards And Aspect Ratios**.

To render your **AVI** movie, display the camera's view on the screen with **View→Camera**, and then select **Render→Render Movie** found on the Menu bar. This will open up the Movie Spec Editor for you to fill in your Movie Parameters.



Movie Parameters Dialogue Box

Here you will find your choices of:

- **Movie Name**
Here is where you name your movie. The procedure is the same as with naming images (see above).
- **Format**
Here is where you make the choice of which format to use. While it seems as if AVI would be your only option it may be preferable to actually pick one of the image formats (see: image formats above). *Why?* Although you will not be able to see your animation immediately quite often, if you wish to manually put in a series of frames of special effects it is easier to have the frames handy as images. Also, there may be a distortion or artifact that you were not expecting and instead of rendering a whole movie again it

would be more efficient to just render the affected image/images after a small tweak if needed. Since movies are basically just a series of images run together to create the illusion of motion movie editors are set up to handle images and using an image format is no problem. You might actually be able to do time adjustments for individual frames to create effects like 'slow motion'. If you use an image format instead of AVI, when it is saved it will take your movie name that you entered and use that with sequential numbering after.

- **Image**

Along with AVI you will find your graphical image selections. These are described above under *Image File Format Editor*. Your movie format and resolution should already be adjusted to the movie parameters you previously selected. If you have not yet you can still do this in the scene editor. See **Chapter 7: SceneEditor**. Otherwise put your resolution choice in the boxes. The numbers are in pixels.

- **Znear/Zfar**

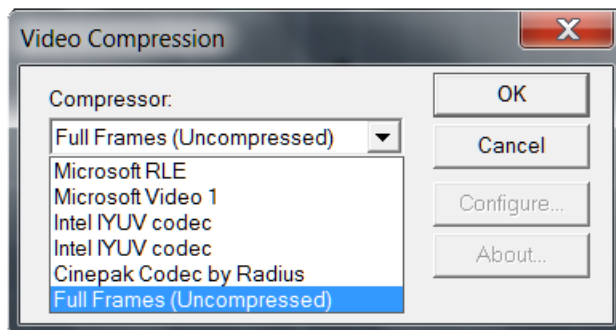
As above

- **Frames**

You have the choice of doing the whole AVI at once by choosing ALL. If it is a longer animation you may wish to break it up into smaller chunks within a specific frame range. Breaking it up also can help speed up render times if you have your own home-version Render Farm. A Render Farm is basically more than one computer each rendering their own specified range of frames but at the same time as each other. By having copies of the same version of Anim8or and your project on each computer you will be able to have them run your rendering. Another reason you may want only a small group of frames is to replace a portion of your movie that had problems with a re-rendered version.

When you have concluded your choices click on **OK**.

The **Video Compression** dialogue box will then open and you will be faced with the options available to you depending on the codecs available on your hardware. By default you also have the option of an choosing Uncompressed AVI. This is usually your best choice, especially if you plan on using a separate movie editor later to add other AVIs, sound, special effects, or transitions. Unlike codecs which must be on the machine that will display the animated images this option should be viewable by most. There are a multitude of various codecs so they will not be described here.



Video Compression

After you have made your choice by highlighting it just hit the **OK** button. If you have decided to make an AVI of the Alpha or Depth channel the Video Compression dialogue box will open immediately again for you to make your Alpha or Depth channel AVI codec choice. (AVI first

then Alpha/Depth AVI second) They do not have to be the same. The Depth channel AVI currently is not available however it will default back to an Alpha channel AVI but still have the Depth channel tag.

When it finishes setting up (NO ... it did NOT freeze up on you) you will be able to watch it being rendered frame-by-frame. The current frame number it is working on will be displayed along the bottom.

Remember, it can be cancelled by pressing the **ESC** key on the keyboard. It may take a while.

Unlike the image editor it will automatically save both while it is rendering, the AVI and the Alpha channel AVI with its' tag on the original filename.

Currently, when the movie is finished the render screen will close automatically and take you back to the Scene Editor.

"YOU ARE NOW A MOVIE PRODUCER ... CONGRATULATIONS !"

Appendix A

Keyboard Shortcuts

***Note:** Please keep in mind that Anim8or is in development and that keyboard shortcuts may or may not work as intended. This will be updated as development progresses!*

There are numerous **keyboard shortcuts** that you can use to facilitate your modeling. They do not add new functionality. For every shortcut there is an equivalent toolbar button or menu command that you may use instead.

Commands that are available in more than one editing mode use the same shortcut. The control-style and most lower case shortcuts fall into this category. Some shortcuts perform different command in each editor. These are generally the uppercase ones.

You can also use shortcuts for most menu commands. The name of each shortcut is shown in the menu to the right of the command name.

Menu Shortcuts

The following are standard Windows shortcuts:

^N	New project
^O	Open project
^S	Save project
Sh+^S	Save all

Viewpoint Shortcuts

You can use the numeric keypad to switch the working view to any of the standard viewpoints.

7 Back View	8 Top View	9 Ortho View
4 Left View	5 Front View	6 Right View
1 Camera View	2 Bottom View	3 Perspective View
0 Previous View		. Toggle 1/4 views

Top Toolbar Shortcuts

These shortcuts are for the top toolbar:



^Z	^Y	^X	^W	^F	^U	^A
Undo	Redo	Cut	Wire Frame	Flat Shaded	Smooth Shaded	Material Toolbar
^R	^G	^T	^L	^P	^D	
Arc Rotate	Grid Snap	Fast Select	List Items	Graph Editor	CAS Annotations	

Object Editor Shortcuts

		A Edit mode	V Viewpoint mode
		O Axis mode	P Point edit mode
		w World coordinates	o Object coordinates
		j Screen coordinates	
		x X Axis	y Y Axis
		z Z Axis	
		a Select	d Drag select
		m Move	r Rotate
		n Non-uniform scale	s Scale
		l Line	p Path
		u UV editor	
		s Sphere	c Cube
		y Cylinder	u Primitive mesh
		g N-gon	t Text
		w Warp modifier	

Point Editor Shortcuts














































































































































		A Edit mode	V Viewpoint mode
		O Axis mode	P Point edit mode
			w World coordinates
			o Object coordinates
			j Screen coordinates
			x X Axis
			y Y Axis
			z Z Axis
			e Edge
			g Face
			
			
			
			
			
			
			
			
			
			
			
			
			
			
			
			
			
			
			
			
			
			
			
			
			

Figure Editor Shortcuts

























		A Edit mode	V Viewpoint mode
			
			
			
			
			

		E Edit bones	R Rotate bone
		L Bone length	N New bone
		S Skin object	
		X Show axis	B Show bones
		O Show objects	I Show IK controls

Sequence Editor Shortcuts

		A Edit mode	V Viewpoint mode	
		w World coordinates	o Object coordinates	j Screen coordinates
		x X Axis	y Y Axis	z Z Axis
		a Select	R Rotate bone	
		i Inverse kinematics		
				
Ret	Play	Space	Pause/Resume	Escape Stop
		X Show axis	B Show bones	
		O Show objects	I Show IK controls	
		G Show ghosts		
		K Animate		

Scene Editor Shortcuts

		A Edit mode	V Viewpoint mode
		w World coordinates	o Object coordinates
			j Screen coordinates
		x X Axis	y Y Axis
			z Z Axis
		a Select	d Drag select
		m Move	r Rotate
		s Scale	p Edit path
		R Rotate bone	i Inverse kinematics
	Ret Play	Space Pause/Resume	Escape Stop
		X Show axis	B Show bones
		O Show objects	I Show IK controls
		O Show camera	I Show path
		K Animate	

Appendix B

Standards and Aspect Ratios

Image Standards

Graphic Image Standards

There are a multitude of various image standards however only the three used in Anim8or have been described briefly below to give you an idea of the proper one for you depending on your final use. If you are unsure it is always a safe bet to use the uncompressed **.bmp** standard because it can be converted to other standards that you may require.

- **.BMP (Bitmap)** *Anim8ors' default : 72 pixels/inch*
Bitmap is the **uncompressed** standard for raster graphic images. A raster image is basically just a finite 2D indexed array (a grid) that holds the color information for each pixel of the complete image. Using various number of bits to determine the values it can be transferred, edited, converted to any other standard, and saved free from the worry of having problems and distortions. Its' one setback is since it is uncompressed the file sizes can be massive depending on the image size and the number of bits involved to determine the colors. While bitmaps do not have an alpha they can themselves be used for an alpha channel as well as making excellent height maps and various other rendering aids such as specular maps.
- **.JPG (Joint Photographic Experts Group)** *Anim8ors' default : Quality 90 %*
JPEG is a **lossy** standard that is widely used for photographs and raster graphic images. It uses the RGB (*RedGreenBlue*) color space but for professional printing it easily transfers over to the CMYK (*CyanMagentaYellowBlack*) color space. While it delivers excellent quality for the end product and it provides excellent compression it is not suited for multiple editing because the degradation of the quality of the image increases with each edit. This works best with realistic or photorealistic images that are limited in sharp contrast areas that may cause discolorations and artifacts. On the practical side JPEGs are invaluable if you want to make your own hand-made Normal maps.
- **.PNG (Portable Network Graphics)** *Anim8ors' default : RGBA*
PING is currently the most commonly used **lossless** standard of raster images. It includes both 24-bit RGB and 32-bit RGBA (*RedGreenBlueAlpha*). This standard was never intended to be used for printing when it was developed and therefore it does not support CYMK. It is an excellent choice for working with and saving drawings, line art, sharp contrast graphics, and images that require multiple editing. It also gives you the option of having transparency in your image.

Film Standards

In the history of cinematography there have been many formats and standards tried and developed. Many have changed our lives and will live on forever while others passed in the night after giving us a brief glimpse of something unique or what the future may hold for us. What

they have been a part of in what has been produced has become parts of our heritages, dreams, and have given us new ideas for the future. Below you will find brief descriptions of a few of the standards that have developed into what we use today but they are far from all of them or their variations. Delve into their stories and you may find them fascinating. Notably missing from this list are the stories of 70 mm film, MGM Panavision, Todd-AO (2.21:1 at 24), and the exciting Cinerama that used three 35 mm cameras (then three projectors) simultaneously side-by-side to produce and project the film (2.59:1 at 26 frames per second). An integral part of the history of the industry is the world-wide association of professionals from many aspects of the business that came together and formed The **Society of Motion Picture and Television Engineers** (SMPTE) in 1916 to create the standards we all use today.

The descriptions below include the basics that will help you decide what to use when it is time to render your masterpiece in Anim8or. Hopefully, should you decide to go forward in your endeavors it may even steer you in the right direction of where to go next.

Definitely, ENJOY yourself because if you aren't then it isn't worth doing!

35 MM Film

*Do not confuse 35 mm movie film with 35 mm picture film. While using the same sized film, the perforations and dimensions of the area to be exposed are different. 35 mm picture film **has** been used in the movie industry by using stop-motion frame by frame exposures to produce special effects and animations.*

- **Silent Film Standard**

35 mm

Aspect Ratio: 1.333:1 (4:3)

Frame rates: 16, 12, and variable frames/second

While there were many formats and film dimensions used in the earliest years eventually the 35 mm "Silent Film Standard" developed in 1892 won out with the 4:3 aspect ratio becoming officially adopted in 1917. Framerates changed to finally becoming 16 frames/second. That was used because it was the slowest rate that enabled nearly smooth motion while viewing. However the earlier ones quite often had variable frame rates that were dependent on the speed at which the cameraman turned the manual hand-crank of the camera. Tiredness and distractions caused many a movie to be "visually interesting". (Hint for Anim8or users: You might like to utilize this fact in your movies if you want to re-create a "silent" type of movie of your own.)

- **Talkies (pre-Academy)**

35 mm

Aspect Ratios: 1.333:1 (4:3)

Frame rate: 24 frames/second

When sound was introduced it wasn't like today. Informally called the "Talkies", these films had their sound placed alongside the image on the film inside the perforations on one side. The addition of this caused the image area to be changed to an aspect ratio of 1.19:1, causing many problems with the movie projectors used at the time. Pictures seemed distorted to viewers and so a new standard taking this into account was developed and came into use in the early 1930's. This new standard was called the Academy standard.

- **Academy - Academy of Motion Picture Arts and Sciences**
35 mm
Aspect Ratios: 1.333:1 (4:3)(image)
Frame rate: 24 frames/second
The *Academy Ratio* of 1.375:1 was adopted so that after sound was included on the film the image would retain the 1.333:1 (4:3) ratio. It was used until the early 1950's when the new Widescreen standards started to replace them but even so, it is still in use today.
- **CinemaScope**
35 mm
Aspect Ratios: 2.35:1
Frame rate: 24 frames/second
Even tho the process was invented in France in 1926 it wasn't until 1953 when Cinemascope started using specially designed anamorphic lenses to create anamorphic films. Anamorphic lenses took the image and shrunk it down (distorted) onto 35 mm film. Later by using other lenses this process was reversed to restore the original sized image for viewing. Initially, the aspect of the image was 2.66:1 but then when sound was added the image aspect ratio became what is considered the modern anamorphic classic standard to 2.35:1.
(Anim8or's Movie Image Parameters dialogue box labels this as the 2.35:1 Cinematic Panorama)
- **Paramount VistaVision**
35 mm
Aspect Ratios: 1.66:1, 1.85:1 (and) 1.85:1, 2:1 (depending on how it was filmed and projected.)
Frame rate: 24 frames/second
Almost a year after CinemaScope came out **Paramount** developed its own widescreen format known as "VistaVision". However unlike the other, VistaVision was a non-anamorphic filming standard. They accomplished this by repositioning the film so it was shot horizontally so as to produce a larger area for the image. Depending on how it was filmed and processed it was able to be projected in one of the available ratios (1.66:1, 1.85:1, or 2:1). Two of these Paramount ratios came to be commonly used. The 1.85:1 was mostly used in the United States while Europe went with the 1.66:1 standard.
(Anim8or's Movie Image Parameters dialogue box labels this as the 1.85:1 Cinematic Widescreen)
- **Techniscope**
35 mm
Aspect Ratios: 2.35:1, 2.39:1 (after 1970, still one of the top aspect ratios used)
Frame rate: 24 frames/second
Coming to life in 1960 in Italy the Techniscope format was used during the next two decades to produce mainly low-budget films in Europe. Its grainy quality was the result of converting the cheaper format into one that could be distributed for use. Well known are many of the "spaghetti westerns" and "horror flicks" of the era that were filmed using this.

- **Super 35**

35 mm

Aspect Ratios: 2.40:1 (however read the description below)

Frame rate: 24 frames/second

While this format was originally developed in 1956 it wasn't until the early 1980's when it was really started to being used. It involved shooting the film on standard 35 mm film (4:3) and then **cropping** it to produce the 2.40:1 aspect ratio. The introduction of digital editing equipment has allowed this format to be in use again while keeping the originals without changes. It should be noted that the "SuperScope" format was its' forerunner.

16 MM Film

- **Standard 16 mm**

16 mm

Aspect Ratio: 1.37:1

Frame rate: 24 frames/second

Released in the early 1920's "silent" 16 mm film equipment was targeted for home use because it was cheaper than using standard 35 mm. By the mid-1930's the introduction of the availability of sound made the 16 mm format the standard for educational, industrial, and government films. Even tho it was considered a sub-standard medium the 16 mm film industry became standard for low-cost filmmaking and was used extensively in the Second World War, news, and the television industry. 16 mm is still in use today in many areas however the home user, for a more economical alternative to the 16 mm format had, for the most part, already basically switched to the 8 mm format developed during the Depression specifically for their use.

- **Super 16 mm**

16 mm

Aspect Ratio: 1.66:1, 2.35:1 widescreen (with special lenses developed in 2009)

Frame rate: 24 frames/second

Super 16 was developed in 1969 with the use of single perforations on one side and special modifications to the camera to simulate the Paramount format (1.66:1).

- **Ultra 16**

16 mm

Aspect Ratio: 1.85:1

Frame rate: 24 frames/second

Ultra 16 was invented in 1996 by changes to the camera that allowed for more coverage area over the regular 16 mm film. This was an economical way to avoid the costs and problems of changing standard 16 mm cameras to the Super 16 format.

8 MM Film

- **Standard 8 mm**

8 mm

Aspect Ratio: 1.32:1 (filmed), 1.33:1 (4:3) (projected)

Frame rates: 16 frames/second

8 mm film was developed in the 1930's Depression Era and was released in 1932. Also known as "Regular 8" and "Double 8" the 8mm film was actually 16 mm film. It was used on one side and then the spool was changed which reversed the film to be run through again to expose the other half. After splitting the film in half it was then spliced

together to create the full-length film. It was developed for use for the general public to produce their own home movies.

- **Straight 8**

8 mm

Aspect Ratio: 1.32:1 (filmed), 1.33:1 (4:3) (projected)

Frame rates: 16 frames/second

"Straight 8" single full-length was available in 1935 but temporarily lost out to the marketing for the "Double 8".

- **Super 8 mm**

8 mm

Aspect Ratio: 1.33:1 (4:3)

Frame rates: 24 (professional), 18 (amateur) frames/second

Released in the mid-1960's this format became popular with home moviemakers because it's film was in cartridges that did not require any manipulation as needed in the earlier version. Sound was introduced onto this format in 1973. This is still used during amateur film-making when attempting to recreate various old fashioned looks and styles.

Television Formats

- **SDTV - Standard Definition Television (also known as SD)**

NTSC and PAL format: 480i

Used In: N. America, most of S. America, and Japan

Aspect Ratio: 4:3, 16:9

Frame rates: 24, 30 (interlaced), and 60 (progressive) frames per second

Resolution: 704 x 480 (720x480 (full frame))

Signal Type: Analog and Digital

PAL and SECAM format: 576i

Used In: Europe, Asia, Africa, Australia, and parts of S. America and Oceania

Aspect Ratio: 4:3, 16:9

Frame rates: 24, 25 (interlaced), and 50 (progressive) frames per second

Resolution: 704 x 576 (720x576 (full frame))

Signal Type: Analog and Digital

In SDTV broadcasting, irregardless of the actual size of the image and its aspect ratio, only the centre 704 (horizontal) pixels are used for the transmitted signal (after they are processed).

- **EDTV - Extended Definition Television (also known as Enhanced Definition Television)**

NTSC format: 480p

Used In: television, internet and gaming consoles

Aspect Ratio: 4:3, 16:9

Picture Resolution: 852 × 480

Frame rates: 30 or 60 frames per second

Signal Type: Analog and Digital

PAL format: 576p
Used In: television, internet and gaming consoles
Aspect Ratio: 4:3, 16:9
Picture Resolution: 852 × 480
Frame rates: 25 or 50 frames per second
Signal Type: Analog and Digital

EDTV defines the format that gives a quality between SDTV and HDTV standards. Its unusual resolution is not considered a standard however for television transmission the picture is *scaled* to the proper appropriate proportions.

- **HDTV - High Definition Television**

Formats:

720p	HD Ready (progressive)	16:9	1280×720	
	XGA	4:3	1024×768	
	WXA	16:9	1366×768	
	HD Internet Video	16:9	1280×720	
1080i	Full HD (interlaced)	16:9	1920×1080	Standard HDTV resolution
1080p	Full HD (progressive)	16:9	1920×1080	Standard HDTV resolution

While true Digital HDTV does not require the use of framerates, to ensure backward compatibility for older broadcasting equipment and the ability to use older formats, these standards are still incorporated into HDTV. Analog HDTV itself, in the beginning, had inherent problems and only Japan was able to make it commercially viable.

Frame Rates used for HDTV:

NTSC

Aspect Ratio: 4:3, 16:9
Frame rates: 30 frames per second
Signal Type: Analog and Digital

PAL and SECAM

Aspect Ratio: 4:3, 16:9
Frame rates: 25 frames per second
Signal Type: Analog and Digital

MUSE format: (*Obsolete*)

Aspect Ratio: 5:3
Frame rates: 30 frames per second
Signal Type: Analog and Digital

Analog Standards

The three major Analog standards are NTSC, PAL, and SECAM with the Japanese MUSE no longer used. SECAM is currently being transitioned out, however, NTSC and PAL standards are still much in use and will be for the foreseeable future. Their standards are being used in the new digital standards and conversions are easily possible.

NTSC - National Television System Committee

Used In: N. America, Mexico, South Korea, and Japan

Frame rates: 29.97 frames/second

Utility Rate: 60 Hertz

Format: interlaced(i)

720 x 480 16:9

720 x 486 16:9

864 x 486 16:9

720 x 540 4:3

N. America has now turned to the ATSC digital standard for broadcasting. (Note that there is also a color standard called NTSC. They are not to be confused as they are not the same thing.)

- **PAL - Phase Alternating Line**

Used In: Europe, Asia, Australia, and much of Oceania.

Frame rates: 25 frames/second

Utility Rate: 50 Hertz

Format: interlaced (i)

720 x 576 16:9

768 x 576 4:3

1024 x 576 16:9

- **SECAM - Séquentiel Couleur à Mémoire** (translated "Sequential colour with memory")

Used In: France, Russia, and parts of Africa. (becoming obsolete)

Frame rates: 25 frames/second

Utility Rate: 50 Hertz

Format: interlaced(i)

It is currently being phased out because Europe is changing to the DVB-T digital standard for television.

- **MUSE - Multiple sub-Nyquist Sampling Encoding**

Used In: Japan (now obsolete for the most part)

Frame rates: -

Utility Rate: 60 Hertz

MUSE was considered briefly for use in Europe however the costs of converting from 60 Hertz to 50 Hertz was the deciding factor against it. Japan started using NTSC and is now converted over to a Digital standard.

Digital Standards

- **ATSC - Advanced Television Systems Committee**

Used In: N. America, Mexico, and South Korea.

Aspect Ratio: 4:3, 16:9

Uses: SDTV formats, HDTV formats, 480i, 576i, 720p, 1080i, and 1080p

Display Sizes: NTSC (720 x 480) PAL and SECAM (704 x 576) HDTV (1280 x 720) (1920 x 1080)

Frame Rates: 23.976, 24, 29.97, 30, 59.94, and 60 frames/second (25 and 50 are **not** used).

- **ISDB-T - Integrated Services Digital**
Used In: Japan, S America, and the Philippines.
Aspect Ratio: 16:9
Uses: SDTV formats
Be aware that the full ISDB standard is considered strictly for use on the digital software and hardware platforms and utilizing it in any analog situation (whether by distribution or by using analog devices violates its' " Copyright ") and there are **strict conditions** on copying and use.
- **DVB-T - Digital Video Broadcasting-Terrestrial**
Used In: Europe, most of Asia, Africa, Australia, and parts of S. America and Oceania.
Uses: SDTV formats, HDTV formats
Note that most areas have their own variations of this standard that need to be checked on for specific purposes.
- **DTMB - Digital Terrestrial Multimedia Broadcast**
Used In: China and Hong Kong
Uses: SDTV formats, HDTV formats
It is being considered for use or being used in Cuba and parts of the Middle East.

Aspect Ratios and Resolutions

Having nothing to do with the actual size of an image the *Aspect Ratio* is merely the proportions they have in relation to each other. Written as **width:height** . They allow images to be scaled up or down and keep their original content without distortion. So if you want the aspect ratio of 4:3 the image resolution might be written 640 x 480. The same image without distortion could also be a larger resolution of 1280 x 960.

Paper

Pixels and Dots

Pixels are not Dots despite being erroneously called that by many people. While pixels refer to the single smallest element of a digital raster image, dots refer either to single elements of an image when scanning or to the density of the ink placement of a physical picture when printing. When you scan an image each scan dot ends up becoming one pixel. However, when you print a picture this relationship might be different depending on your hardware, software, and of course your settings. (For anyone interested; dots are also single ink spots used in pointillism, a small round spot, tiny speck, a child, a mark made with a pointed object, the short sound in Morse code, a wisp of cloth or straw, and of course ... a nipple. HONEST it's true ... I am not making it up! It IS in the dictionary ... a real one I mean.)

The direct correlation between Pixels-per-Inch (**PPI**) and Dots-per-Inch (**DPI**) then are shown not necessarily to be true.

The most used PPI/DPI (Pixels-per-Inch/Dots-per-Inch) settings are **72 PPI, 96 PPI, 120 PPI, 300 PPI**. 72 PPI was based on the first CRT monitors. Professional printers usually ask for your image in 300 DPI. However be careful in this. You must make **both the DPI AND Image Resolution** (usually in Inches) available to the printer to obtain the proper results. If you don't

make both known you may end up with either a picture that is severely cropped or one that you need a flatbed truck to haul it away in.

If you will be making an image that you plan on sending to a printer you can determine its' resolution with the formula : (the number of inches of your image) x (the number of pixels per inch) . Example: a 5" x 7" image with 300 pixels-per-inch would be (5)x(300) by (7)x(300) or an image with the pixel resolution of 1500 x 2100.

If you already have an image at a specific aspect ratio and you want to convert it to a different one then the first thing you must do is use the above formula to determine the resolutions of the image that you have and the one you want. Increase or decrease the image you already have in its' own aspect ratio until the image resolution is just above the image and its' resolution that you want. This will give you the closest print possible and probably without cropping or distortions. A little bigger is definitely better than a little too small.

If you are printing it yourself then the choice is yours. While scanning, the more dots-per-inch that you use the better the quality of your digital image (and the larger the file size) but this is not always true in the opposite case. You may find when printing that a lower DPI will give you more than adequate quality for what you want.

Common paper sizes

portrait (vertical), landscape (horizontal)

1 inch = (2.54 cm) or (25.40 mm)

A4	210 x 297	mm	1.4143
Letter	8 ½ x 11	inches	1.2941
Legal	8 ½ x 14	inches	1.6471
Ledger (Tabloid)	11 x 17	inches	1.5455
Crown	15 x 20	inches	1.3333
Royal	20 x 25	inches	1.25

Common Print Sizes

2 1/2 x 3	inches	7:5 (1.4:1)	wallet print size
4 x 5	inches	5:4 (1.25:1)	common print size
4 x 5	inches	5:4 (1.25:1)	Polaroid Instant Peel-Apart
4 x 6	inches	3:2 (1.5:1)	common print size
5 x 7	inches	7:5 (1.4:1)	common print size
6 x 8	inches	4:3 (1.33:1)	
8 x 10	inches	5:4 (1.25:1)	common print size
8 x 12	inches	3:2 (1.27:1)	
11 x 14	inches	14:11 (1.27:1)	common print size
11 x 17	inches	17:11 (1.55:1)	
12 x 15	inches	5:4 (1.25:1)	

Common Resolutions

- **1:1 (1:1) :**
1 x 1 (smallest resolution), 32 x 32, 64 x 64, 128 x 128, 256 x 256, 512 x 512, 1024 x 1024, 2048 x 2048, 4096 x 4096
- **3:2 (1.5:1) :**
480 x 320, 512 x 342, 960 x 640, 1280 x 854, 1440 x 960

- **5:4 (1.25:1) :**
400 x 240, 640 x 512, 1280 x 1024, 2560 x 2048
- **5:3 (1.6667:1) :**
400 x 240, 800 x 480, 1280 x 768
- **4:3 (1.333:1):**
160 x 120, 320 x 240, 640 x 480, 800 x 600, 1024 x 768, 1152 x 864, 1280 x 960,
1400 x 1050, 1600 x 1200, 2048 x 1536, 3200 x 2400, 4096 x 3072
- **16:9 (1.778:1):**
480 x 270, 640 x 360, 960 x 540, 1024 x 576, 1280 x 720, 1366 x 768, 1600 x 900,
1680 x 945, 1920 x 1080, 2048 x 1152, 2560 x 1440, 3840 x 2160
- **16:10 (1.6:1) :**
320 x 200, 640 x 400, 1280 x 800, 1440 x 900, 1680 x 1050, 1920 x 1200,
2560 x 1600

Film

Listed below are the automatic settings used by Anim8or for film:

4:3 NSTC, Academy (image area)
16:9 HDTV
1.85:1 Cinematic Widescreen
2.35:1 Cinematic Panorama

If you plan on using a different set aspect ratio than these remember that Anim8or v1.01.1318 currently *will not* automatically change your 'Other' resolution. You will have to place your own numbers in.

- **VGA**

4:3	640 x 480
16:9	640 x 360
1.85:1	640 x 346
2.35:1	640 x 272
- **PAL 16:9**

4:3	1024 x 768
16:9	1024 x 576
1.85:1	1024 x 554
2.35:1	1024 x 436
- **720p**

4:3	1280 x 960
16:9	1280 x 720
1.85:1	1280 x 692
2.35:1	1280 x 545

- **1080p**

4:3	1920 x 1440
16:9	1920 x 1080
1.85:1	1920 x 1038
2.35:1	1920 x 817

Cinematic 'Big' Screen - Digital Age

When the digital age hit the industry, film was transferred to digital by a process known as telecine. The telecine equipment is also used for post-production work. However, in 2002 the major motion picture giants came together and formed the **Digital Cinema Initiatives (CDI)** to set in place new standards for the Big Screen. It is beyond the scope of Anim8or to produce a new Digital Cinematic Masterpiece but it is still interesting to read to find out where cinema and animation are headed for in the future. Should you decide to head in this direction for your future Anim8or is a GREAT place to start learning.

Displays

The aspect ratios and resolutions listed are based on the various hardware , their standards, and their limitations on displaying them. (Not all hardware is listed below and this is only to be a guide for you if you require them.) When you plan on rendering your animation it would be best if you use the **common** display size unless you have a specific requirement. Where no *common* display size is listed it will depend on your final use.

- **VGA (Video Graphics Array)**

1987 - older CRT (cathode ray tube) displays, still used today in various

640 x 480	4:3	<i>common display size for VGA</i>
640 x 512	5:4	
640 x 360	16:9	
640 x 346	1.85:1	
640 x 272	2.35:1	

[VGA special note: There actually are quite a bit more resolution settings than listed here however they were dependant on settings. Height in lines was determined by the screen refresh rates and width was determined by the color mode (how many colors used) and aspect ratio.]
- **HVGA (Half VGA)**

- older Macintosh computer displays, iPhones, smartphones, PDAs

480 x 320	3:2	<i>common display size for HVGA</i>
480 x 360	4:3	
512 x 342	3:2	Macintosh B/W
512 x 384	4:3	Macintosh Color Classic
640 x 240	8:3	
- **qVGA (quarter VGA)**

*(Please note that **q** stands for **quarter** (1/4) and **Q** stands for **Quad** (4x))*

- Nintendo portables, Apple products, handheld portable devices, cheap cell phones

320 x 240	4:3	<i>common display size for qVGA</i>
320 x 256	5:4	
376 x 240	14:9	

- **HqVGA (Half Quarter VGA)**
- cell phones and PDAs
240 x 160 3:2
- **qqVGA (Quarter Quarter VGA)**
- early webcams and digital cameras
160 x 120 4:3 *common display size for qqVGA*
160 x 144 10:9
- **WqVGA (Wide qVGA)**
- portable DVD players
400 x 240 5:3
480 x 272 16:9
- **DVGA (Double VGA)**
- some newer Apple products
960 x 640 3:2 *common display size for DVGA*
- **WVGA (Wide VGA)**
- mobile phones
854 x 480 16:9 *common display size for WVGA*
768 x 480 8:5
800 x 480 5:3
- **CGA (Color Graphics Adapter)**
1981 - IBM color display monitors
320 x 200 16:5 *common display size for CGA*
640 x 200 8:5
160 x 200 4:5
- **EGA (Enhanced Graphics Adapter)**
1984 - IBM enhanced color display monitors
640 x 350 64:35 *common display size for EGA*
640 x 200 16:5
320 x 200 8:5
- **SVGA (Super VGA)**
1987 (1989 modes) - IBM and compatible computer displays
800 x 600 4:3 *common display size for SVGA*
640 x 480 4:3
640 x 512 5:4
1024 x 768 4:3 later resolution
- **qSVGA (Quarter SVGA)**
400 x 300 4:3 *common display size for qSVGA*
- **WSVGA (Wide SVGA)**
1024 x 600 16:9 *common display size for WSVGA*
- **XVGA (Extended VGA)**
see: SXGA (Super XGA) below

- **8514/A**
 1987 - first high resolution (interlaced). Used in graphic cards, both CRT and LED displays, early laptops, and modern tablets.
 1024 x 768 4:3 *common* display size for 8514/A
 640 x 480 4:3 *common* display size for 8514 (precursor to 8514/A)
- **XGA (Extended Graphics Array)**
 1990 - IBM standard, early Microsoft Windows
 1024 x 768 4:3 *common* display size for XGA
 640 x 480 4:3
 800 x 600 4:3
 960 x 720 4:3
 1280 x 1024 5:4
 1360 x 1024 4:3
 1600 x 1200 4:3 at 50 Hertz screen refresh rate
- **WXGA (Widescreen XGA)**
 - widescreen notebooks (HDTV or HD ready)
 1280 x 800 8:5 *common* display size for XGA
 1280 x 720 16:9
 1366 x 768 16:9
 1280 x 854 3:2
 1280 x 960 4:3
 1440 x 900 16:10
 1440 x 960 3:2
- **QWXGA (Quad WXGA)**
 2048 x 1152 16:9 *common* display size for QWXGA
 1792 x 1344 4:3
- **SXGA (Super XGA)**
 (also known as XVGA (Extended VGA))
 - standard for 17" and 19" monitors
 1280 x 1024 5:4 *common* display size for SXGA
- **WSXGA (Wide SXGA)**
 1440 x 900 8:5 *common* display size for WSXGA
- **SXGA+ (SXGA Plus)**
 1400 x 1050 4:3 *common* display size for SXGA Plus
- **WSXGA+ (WSXGA Plus)**
 1680 x 1050 8:5 *common* display size for WUXGA Plus
 1776 x 1000 16:9
- **UXGA (Ultra XGA)**
 1600 x 1200 4:3 *common* display size for UXGA
- **WUXGA (Widescreen UXGA)**
 - "23" to "27" LCD monitors, and home movie projectors
 1920 x 1200 16:10 *common* display size for WUXGA

- **QXGA (Quad XGA)**
 - iPad (3rd Generation)
 2048 x 1536 4:3 *common display size for QXGA*
- **WQXGA (Wide QXGA)**
 2560 x 1600 8:5 *common display size for WQXGA*

(Display Size Note: There are more newer ones that can be listed but their larger display sizes do not lend themselves to being used in Anim8or easily.)

Interlaced and Progressive

Interlaced (i) and Progressive (p) (non-interlaced) display scans. In earlier days in order to get rid of flickering and produce a smoother display interlacing was developed. Interlacing is merely reading and writing the picture fields (Two fields for each image frame, each consisting of half a picture) of the image. One pass would write the even lines of the picture on the display and the next pass would write the odd lines (This does not affect frame rate). However with the faster screen refresh rates and the development of digital this technique was no longer needed and so progressive writing became used. This would write the display sequentially, one line after another. The inherent problem between the two however is *time*. Each half of an interlaced picture when it is taken has *time* involved so when displayed together the motion appears smooth and clean. Progressive scanning on the other hand does not take *time* into account so that when you convert an interlaced screen image into a progressive screen image you will quite often find distortions such as mismatched pictures and/or horizontal lines in your picture where motion is involved. This has nothing to do with the pictures or the hardware but only the *time* (or lack thereof) factor involved in *i* and *p* displays. (While progressive displays have no problem with the progressive formats, when they use standard interlaced formats it is required that the framerate be doubled to avoid image flickering.)

You may notice that you may see a number after the format symbol. It stands for either the utility rate in Hertz such as 480i60 or the frame rate 576i/25 or 480/30i. If you are wondering how the electrical utility rates affect the standards it is because the hardware was dependent on the type of power being used in an area (60 Hertz in the Americas and 50 Hertz in Europe and so forth).

- **480i**
 interlaced
 60 Hertz areas
 640 x 480 4:3
- **576i**
 interlaced
 50 Hertz areas
 (cropping involved for transmission ... 4:3 in analogue and 16:9 in digital)
 480 x 576
 544 x 576
 704 x 576
 720 x 576
 768 x 576

When progressive scan content is used in 576i the wrong field is read first so it can cause problems by reading fields from two separate images. Care must be used in this situation.

- **576p**
 progressive
 (cropping involved for transmission ... 4:3 in analogue and 16:9 in digital)
 480 × 576
 544 × 576
 704 × 576
 720 × 576
 768 × 576
- **720p**
 progressive (in order to avoid confusion realize that there is **NO** 720i standard)
 standard HD (HD ready) 960 x 720 4:3
 widescreen 1280 x 720 16:9
 notebook computers 1366 x 768 16:9
- **900p**
 progressive
 HD+ (High Definition Plus) 1600 x900 16:9
- **1080i**
 interlaced
 HD 1920 x 1080 16:9
- **1080p**
 progressive
 standard 1440 x 1080 4:3
 HD widescreen 1920 x 1080 16:9
 Full HD+ 2160 x 1080 18:9

Legal Disclaimer: Even tho publicly available facts cannot be copyrighted and Fair Use allows dissemination of information for research and education I would still like to give a **BIG SHOUTOUT** to the researchers and authors who have worked hard and uploaded their content to **Wikipedia.com** under the Public Domain and Creative Commons (many various versions) licenses. It has assisted me in refreshing my old memory, verified various facts, and has filled in missing gaps. Know that your hard work is being put to good use and that it is helping others to achieve their dreams. **THANK YOU ALL !**

Appendix C

an8_format

Filename: an8_format.txt

Date: 21-Sep-03

Format of Anim8or .an8 Files - V0.85

This document describes the .an8 file format used by Anim8or to save project data between sessions.

Overview:

A .an8 file is stored as a text file and can be viewed and edited by a simple text editor such as Notepad. Files are free format much like those used by popular programming languages like C. At the text level they are similarly composed of scanner tokens such as identifiers, numbers, strings and punctuation. Tokens are organized into nested "chunks" that contain specific data used by each specific chunk. Chunks share a common format so that a parser can easily skip over unrecognized or unneeded data. The .an8 format is designed to be both forward and backward compatible to a large degree largely due to chunks.

Tokens:

The basic tokens used by the .an8 format are:

<\$ident> - identifiers similar to those used by C. They may begin with letters and the underscore character "_" and may contain letters, digits, and underscores. Case is important so a capital "A" is different from a lower case "a". No spaces may appear within an <ident>.

<\$int> - integer constants. One or more digits "0" through "9". A basic <\$int> does not include a sign. **<\$float>** - floating point constants. The usual C style floating point numbers are allowed. Again a <\$float> cannot contain a leading sign.

<\$string> - string constants. C style strings beginning and ending with a double quote ". Internal spaces are allowed. The backslash character "\" is used to escape special characters such as a double quote " so they may be used in a string. For example the string:

"He said, \"What do you want?\""

would have the value:

He said, "What do you want?"

Similarly back slashes are escaped by a backslash.

"{" and "}" - braces. Used to enclose the body of a chunk. These never appear anywhere else in a .an8 file (except within a string) and are used as the basic parsing tokens. "+" and

"-" - plus and minus. Used in signed numeric values.

"(", ")", ",", etc. - delimiters. Various special characters are used as delimiters through out.

Chunk Format:

The basic format of a chunk is an identifier, followed by an open brace, the body of the chunk, and a closing brace:

<\$ident> "{" <body> "}"

The format of the body depends on which chunk it is in. It may contain a fixed amount of data, a variable amount, more chunks, or be empty.

Examples of a chunk are:

size { 12 }

color { 0.5, 0.75, 0.33333 }

closed {}

texture { "checks" blend { 1 } }

File Layout:

An .an8 file is one or more chunks. Any specific chunk is optional, and the chunks aren't required to be in any specific order. However it's advisable to include the <header> chunk, and to put it first, to help the Anim8or parser deal with any incompatibles that are introduced between different versions. The order of the top level chunks is:

- <header> - Contains version number and related data.
- <description> - String with user's description of the file.
- <environment> - Frame rate and ground grid settings.
- <texture>* - Description of texture files used.
- <material>* - Global materials.
- <object>* - Objects.

- <figure>*** - Figures.
- <sequence>*** - Sequences.
- <scene>*** - Scenes.

In the syntax descriptions below, items with an asterisk "*" after them may appear any number of times, including none. Those with a plus "+" must appear once but can be repeated. Tokens within square brackets "[" and "]" are optional. Finally things within angle brackets "<" and ">" are described by other syntax rules.

Headers:

=====

<header> :: = header { <version> <build> }

<version> :: = version { <\$string> }

The version string has the form <major> "." <minor> such as "0.85". It can have additional information at the end as well such as "0.85beta"

<build> :: = build { <\$string> }

The build string is used to identify the build number of the Anim8or executable that created the .an8 file.

Description:

=====

<description> :: = description { <\$string>* }

Normally description strings represent a single line of text and are terminated with a <cr> <lf>:

"This is my alien model.\0d\0a"

Anim8or **truncates descriptions to 4096 characters** when reading them.

Environment:

=====

<environment> :: = environment { <grid> <framerate> <limitplayback> }

<grid> :: = grid { <int-const> <float-const> <float-const> <float-const> }

Specifies the modeling grid's behavior. If the first int is non-zero then auto-grid is enabled and the grid size is automatically determined by the view. If it's zero then the three following values set the modeling grid spacing, the Scene editor's grid spacing, and the ground floor's grid size. An of the later values may be omitted and a default value will be used.

<framerate> :: = framerate { <int-const> } Sets the Scene and Sequence playback frame rate. Ignored unless the <limitplayback> chunk also appears.

<limitplayback> :: = limitplayback { }

Limits the playback frame rate to the value in <framerate>, or to a default value if no <framerate> chunk is present.

Texture:

=====

<texture> :: = texture { <name> <invert> <cubemap> <file>+6 }

Define a texture name and bind it to an image file or files. The string <name> is the name used within Anim8or for this texture. <invert> is an optional empty chunk. If present then the image(s) for this texture are inverted top to bottom before they are used. <cubemap> is an optional empty chunk. It indicates that this texture is a cube map. The <file> chunks contain a single string for the name of the files that hold the texture image. Only one <file> chunk is used for normal textures and six <file> chunks are used for cube map textures.

Material:

=====

<material> :: = material { <name> <surface> <backsurface> }

The name of the material is given by <name> and the front and back surfaces are described by the other chunks. Single sided materials don't have the <backsurface> chunk while two sided materials do.

**<surface> :: = surface { <ambient> <diffuse> <specular> <emissive> ...
... <alpha> <brilliance> <phongsize> <map>* ...
... <lockambdiff> }**

<ambient>, <diffuse>, <specular> and <emissive> chunks describe the respective colors and constant weights, and if there is a corresponding texture what it is. Yes, I know <ambient> is misspelled.

<alpha> is a int-const chunk that sets the overall transparency of the material.

255 is opaque and 0 is completely transparent. Values are clamped to this range.

<brilliance> is a float chunk that sets the brilliance factor.

<phongsize> is a float chunk that sets the phong roughness factor.

<map>* is one of multiple chunks that describe a texture map used for things other

than the four colors: ambient, diffuse, specular, and emissive.

Currently supported uses are for transparency, bump map, and environment map.

<lockambdiff> is an empty chunk that, if present, uses the diffuse values for both the diffuse and ambient color properties.

<backsurface> :: = backsurface { ... same values as surface chunk ... }

<ambient> :: = ambient { <rgb> <factor> <texturename> <textureparams> }

<rgb> is the basic color.

<factor> is a float chunk with the ambient weighting factor.

<texturename> is a string containing the name of the texture used in the diffuse color. This is not the file name but the internal name.

<textureparams> describe how the texture is blended with the base color.

<diffuse> :: = diffuse { ---- same as <ambient> ---- }

<specular> :: = diffuse { ---- same as <ambient> ---- }

<emissive> :: = diffuse { ---- same as <ambient> ---- }

<rgb> :: = rgb { <\$int> <\$int> <\$int> }

Three integer values representing the red, green and blue components of a color. They are clamped to 0 to 255, then scaled so to the range 0.0 to 1.0 representing the darkest and brightest values.

<textureparams> :: = textureparams { <blendmode> <alphamode> <percent> }

<blendmode> and <alphamode> specifies how that a texture will be combined with the base color. <percent> holds a single signed integer percentage that is used to blend the overall strength of the texture. If it is 100 then the combined result is used as the final color. If it is 50 percent then the combined color is averaged with the base color. For bumpmaps <percent> is the strength of the bumps and can go from -100 to 100 percent.

<blendmode> :: = blendmode { [decal darken lighten] }

The blendmode chunk holds a single identifier from the list above. darken means that the texture's color is multiplied by the base color, decal means that the texture color replaces the base color, and lighten adds the texture color.

<alphamode> :: = { [none layer final] }

The alphamode chunk also holds a single identifier. None means that the alpha component is ignored, layer blends the texture color with the base color proportional to the value of the alpha component, and final uses the texture's alpha component as the transparency value for the final material color. Note: alphamode only applies to the diffuse color and is overridden by the transparency texture if it is present.

<map> :: = map { <kind> <texturename> <textureparams> }

The <map> chunk defines a texture image usage that is used for things other than a simple color, such as in the ambient, diffuse, specular, or emissive attributes. The <kind> chunk says how it will be used. The <texturename> chunk names which texture it is, and the <textureparams> specify the strength, etc.

<kind> :: = kind { <\$string> }

The <\$string> parameter can be "transparency", "bumpmap" or "environment", or it can be one of the four color textures "ambient", "diffuse", "specular", and "emissive". Other values are ignored. If a color texture is specified in both a color component such as the <diffuse> chunk and in a <map> component the last one in the file will be used.

Object:

=====

<object> :: = object { <name> <material>* <component>* }

The object's name is given by <name>. <material> chunks in an <object> chunk are local to that object and cannot be used in other objects. There can be any number of <material> and <component> chunks in any order. Normally the <material> chunks are output first by Anim8or.

<component> :: = [<mesh> <sphere> <cylinder> <cube> <subdivision> <path> <textcom> <modifier> <image> <namedobject> <group>]

A <component> is any one of the listed chunk types. All <component> types of chunks have a <name> string as the first value. They also may have both a <base> and a <pivot> chunk. <namedobject> components can only appear inside of a <bone> chunk as part of a <figure>.

<base> :: = base { [<origin>] [<orientation>] }

A <base> chunk can have either an <origin>, an <orientation>, or both. This chunk gives the location and orientation of the component relative to it's parent. If the component is in a group then the group is it's parent, otherwise it is the Object.

<pivot> :: = pivot { [<origin>] [<orientation>] }

The <pivot> chunk specifies the location and orientation of the pivot, the coordinate system used when you interactively scale and rotate a component in the Object editor.

<origin> :: = origin { <point> }

This chunk represents the location of the component. If not present then the component is located at (0, 0, 0).

<orientation> :: = orientation { <quaternion> }

This chunk holds the orientation of the component as a quaternion. If not present the it is unrotated, with the X coordinate pointing to the right, the Y pointing up, and the Z pointing out of the screen in a front view of an object. For a description of the wonders of quaternions, see a good graphics text, or the original paper by Ken Shoemaker.

```

<mesh> :: = mesh { <name> <base> <pivot> <material> <smoothangle> ...
                ... <materiallist> <points> <normals> <edges> ...
                ... <texcoords> <faces>

```

<material> is the material used for new faces when they are added to a mesh. Even though a <material> chunk may contain a full material description, only the name component is used here. The material should be defined elsewhere, either in the global materials of the Object's local materials.

```

<smoothangle> := smoothangle { <float-const> }

```

This is the threshold angle in degrees that determines if an edge is as a crease or is smooth. Angles larger than the threshold are shown as creases or corners; those equal or smaller are shown as smooth. The default value is 45 degrees.

```

<materiallist> :: = materiallist { <materialname>* }

```

This is a list of material in a Mesh. They are numbered from 0. Material 0 is the default material.

```

<materialname> :: = materialname { <$string> }

```

The value of <\$string> is a material name. It should be defined either as a global material or as a material local to this Object.

```

<points> :: = { <point>* }

```

These points are the basic vertices used to form the Mesh. They are numbered from 0.

```

<normals> :: = { <point>* }

```

The normals are stored as unit length vectors. This chunk is not usually present and is ignored by Anim8or. It will be output if the Options->Debug->OutputNormals flag has been set for use by other applications.

```

<edges> :: = { <edge>* }

```

A list of edges. Edges are not normally output by Anim8or as this information is already present in the <facedata> chunk. When an edge either 1) isn't used in any face, or 2) has some special properties set by the user, it will be listed here. Optionally, all edges can be output with the Options->Debug->OutputEdges flag.

```

<edge> :: = ( <$int> <$int> [ <int-const> ] )

```

The first two values in an edge are indices into the points array for the ends of the edge. The index of the first point is always less than that of the second. The third value, if present, is the user set sharpness value. If this Mesh is subdivided then this edge will not be smoothed with the adjacent edges for this number of times, creating a relative sharp corner in the surface. If the value is -1 then it is always shown as Creased despite the actual angle it makes.

```

<texcoords> :: = texcoords { <texcoord>* }

```

The UV values used for texture coordinates. These are indexed starting at zero.

```

<texcoord> :: = ( <float-const> <float-const> )

```

The values represent the U and V texture coordinates.

<faces> :: = faces { <facedata>* }

One set of <facedata> data is listed for each face.

<facedata> :: = NUMPOINTS FLAGS MATNO FLATNORMAL ...

... ((POINTDATA-1) (POINTDATA-2) ... (POINTDATA-N))

NUMPOINTS is an <int-const> giving the number of sides in this face. It must be at least 3. Each face can have a different number of sides.

FLAGS is an <int-const> with each bit as a flag describing what kinds of data this face has, and other properties of the face:

1 - SF_SHOW_BACK - Don't backface cull this face.

2 - SF_HAS_NORMALS - This face has normal data in this file.

4 - SF_HAS_TEXTURE - This face has texture coordinates.

MATNO is an <int-const> for which material to use on this face. It is an index into the materials listed in the <materiallist> chunk for this Mesh.

FLATNORMALNO - The index of the faces normal in the normal array for this Mesh. This value is normally set to -1 indicating that there is no face normal stored in the file.

POINTDATA - This is one or more <int-const>s, depending on the value of FLAGS. The first value, which is always present, is an index into the <points> array for the location of the vertex. If the SF_HAS_NORMALS bit is on then it is followed by the index on the point's normal in the <normals> array. Note: this normal data is not used by Anim8or. It can optionally be exported, however, to help other programs use Anim8or data more easily. Finally if the SF_HAS_TEXTURE bit is set then a texture coordinate is present as the last value. For example:

```
faces {  
    3 5 0 -1 ( (2 6) (0 5) (1 7) )  
    3 5 2 -1 ( (3 8) (0 5) (2 3) )  
    4 5 2 -1 ( (4 9) (5 10) (1 7) (2 6) )  
}
```

shows a Mesh with 3 faces. The first two have 3 sides and the third one has 4. They all have texture coordinates (bit 4 is set in the flags field) and no face normals (the face normal value is -1). The first face uses material number 0 and the other two use material number 2.

**<sphere> :: = sphere { <name> <base> <pivot> <material> ...
... <longlat> <geodesic> }**

A <sphere> has either a <longlat> or a <geodesic> chunk, but not both.

<longlat> :: = longlat { <\$int> <\$int> }

The two integers are the number of divisions used to make the Sphere in the vertical and horizontal directions.

<geodesic> :: = geodesic { <\$int> }

A geodesic sphere is built from a octahedron whose faces are divided into smaller triangles. Each original edge is divide into <\$int> edges and the faces are split accordingly into smaller triangles. The vertices are then projected onto a Shpere.

**<cylinder> :: = cylinder { <name> <base> <pivot> <material> <length> ...
... <diameter> <topdiameter> <longlat> ...
... [<capstart>] [<capend>] }**

<length> :: = length { <\$float> } This is the length of the cylinder,

<diameter> :: = diameter { <\$float> } This is the diameter of the start of the cylinder. It is also the diameter of the end if the <topdiameter> chunk is not present. If one of thee values is 0 then that end of the cylinder is collapsed into a single point resulting is a Cone.

<topdiameter> :: = topdiameter { <\$float> }

The diameter of the end of the cylinder.

<capstart> :: = capstart { }

<capend> :: = capend { }

Cap the start and/or end of the cylinder if these chunks are present.

<cube> :: = cube { <name> <base> <pivot> <material> <scale> <divisions> }

<scale> :: = { <\$float> <\$float> <\$float> } The dimensions of the cube in the X, Y and Z directions.

<divisions> :: = { <\$int> <\$int> <\$int> }

The number of divisions along the X, Y and Z axis.

**<subdivision> :: = subdivision { <name> <base> <pivot> <material> ...
... <smoothangle> <working> <divisions> ...
... <materiallist> <points> <normals> <edges> ...
... <texcoords> <faces> }**

A <subdivision> chunk is similar to a <mesh> chunk but has two additional sub-chunks that control how many times it will be subdivided.

<working> :: = working { <\$int> }

The number of times a that this component will be subdivided in the working views.

$$\langle \mathbf{divisions} \rangle ::= \text{divisions } \{ \langle \text{\$int} \rangle \}$$

The number of times a that this component will be subdivided in the rendered views. This value is currently not used by Anim8or.

$$\langle \mathbf{path} \rangle ::= \text{path} \{ \langle \text{name} \rangle \langle \text{base} \rangle \langle \text{pivot} \rangle [\langle \text{extendable} \rangle] \langle \text{bezier} \rangle^* \}$$

A <path> is built from one or more Bezier splines. Multiple splines can be used to cut holes or to define separate, unconnected regions.

$$\langle \text{bezier} \rangle ::= \text{bezier} \{ [\langle \text{closed} \rangle] \langle \text{knot} \rangle^* \}$$
$$\langle \text{closed} \rangle ::= \text{closed } \{ \}$$

If this chunk is present the first and last knots in the spline are connected to form a closed loop. Otherwise they are not.

$$\langle \mathbf{knot} \rangle ::= \text{knot} \{ \langle \text{point} \rangle \langle \text{point} \rangle \langle \text{point} \rangle [\langle \$int \rangle] [\langle \text{corner} \rangle] \}$$

The first <point> is the location of the knot. The next two are the forward and reverse direction vectors. If present the <\$int> value is the number of segments in the segment that begins with this knot. If not then the number is automatically determined when it is used.

$$\langle \mathbf{corner} \rangle :: = \mathbf{corner} \{ \}$$

If this is present the knot is a corner knot and can have a discontinuous first derivative. If it is not present then the values of the forward and reverse direction vectors should be the negation of each other.

$$\langle \text{textcom} \rangle ::= \text{textcom} \{ \langle \text{name} \rangle \langle \text{base} \rangle \langle \text{pivot} \rangle \langle \text{string} \rangle \langle \text{typeface} \rangle \dots$$

$$\langle \text{size} \rangle [\langle \text{bold} \rangle] [\langle \text{italic} \rangle] \}$$

A <texcom> is a True-Type string represented by one or more splines.

$$\langle \text{string} \rangle ::= \text{string} \{ L\langle \$\text{string} \rangle \}$$

This is a Unicode string. Values outside of the 7 bit printable ASCII character set are escaped by a backslash "\" and are written as either 3 or 6 octal digits representing the value of the Unicode character. Six octal digits are used if either the value is greater than decimal 255 or the following letter is a digit. When reading these strings in the digits following a back slash are read and converted into an octal value until either the next character is not an octal digits or 6 digits have been read. For example, the Euro currency symbol could be represented by:

L"\020254"

since it has a hexadecimal value of 0x20ac which is the octal value of 020254. Note: Unicode symbols are not supported in Win95, Win98, WinSE or WinME.

$$\langle \text{modifier} \rangle ::= \text{modifier} \{ \langle \text{name} \rangle \langle \text{base} \rangle \langle \text{pivot} \rangle \langle \text{length} \rangle \langle \text{diameter} \rangle \dots$$

$$\dots \langle \text{segments} \rangle \langle \text{method} \rangle [\langle \text{component} \rangle] \}$$

A <modifier> chunk defines the size, location and kind of a modifier and any <component> that it is bound to.

<segments> :: = segments { <\$int> }

The number of vertical segments that the modifier is divided into.

<method> :: = method { <\$string> <parameter>* }

The kind of modifier. <\$string> names the modifier function and the <parameters> define how it behaves. For example a taper modifier with a strength of 0.5 would look like:

```
method { modifier "taper"
    parameter { "taper" 0.5 }
}
```

since it has a single parameter.

<parameter> :: = parameter { <\$string> <\$float> }

A named parameter to a modifier. <\$string> is the name of the parameter and <\$float> is its value.

<image> :: = image { <name> <\$string> <base> <pivot> <size> }

A reference image. The <\$string> value is the name of the file with the image, or an empty string if it is undefined.

<size> :: = size { <\$int> <\$int> }

The int values are the size of the image in x and y dimensions.

**<namedobject> :: = namedobject { <\$string> <name> <base> <pivot> ...
... <material> <weightedby>* }**

A <namedobject> is a reference to an <object> from within another part of Anim8or, such as a <figure>. <material> is the default material for this object but is currently ignored by Anim8or. <namedobject>s can only appear inside a <bone> definition.

<weightedby> :: = weightedby { <\$string> }

This object's shape is determined partially by the bone <\$string>'s position in this figure.

<group> :: = group { <name> <base> <pivot> <component>* }

A <group> is a collection of zero or more <components>.

Figure:

=====

<figure> :: = figure { <\$string> <material>* <bone> }

A figure. <\$string> is the name of the figure and <bone> is the root bone, which contains all other bones. Any defined materials are local to this figure.

```

<bone> :: = bone { <$string> <length> [ <diameter> ] [ <orientation> ] ...
                ... [ <locked> ] <dof>* [ <influence> ] ...
                ... <component>* <bone>* }

```

<length> is the length of the bone.

<diameter> is the relative diameter of the bone when it is shown in the working views.

<locked> :: = locked { } If this chunk is present then the bone joint cannot be edited.

```

<dof> :: = dof { <$string> <float-const> <float-const> <float-const> ...
                ... [ <locked> ] [ <unlimited> ] }

```

This chunk defines a degree of freedom for this bone. There can be up to 3 instances of this chunk in a bone. The value of <\$string> is the axis and must be "X", "Y", or "Z". The three <float-const>s are the minimum, default, and maximum angles that this angle may take on. If the <locked> chunk is present then this axis of rotation cannot be edited. If the <unlimited> chunk is present then there are no limits to this axis's rotation angle.

```

<unlimited> :: = unlimited { }

```

```

<influence> :: = influence { <float-const> <float-const> <float-const> ...
                            ... <float-const> <float-const> <float-const> }

```

If this chunk is present then the bone has a defined influence region and may be used in skinning. The size floating point values are:

center0 - The location along the bone of the center of the lower end of the influence volume.

inRadius0 - The radius of the inner layer at center0.

outRadius0 - The radius of the outer layer at center0.

center1 - The location along the bone of the center of the upper end of the influence volume.

inRadius1 - The radius of the inner layer at center1.

outRadius1 - The radius of the outer layer at center1.

These six values define two hierarchical "capsules", or cylinders with hemispherical caps on the ends. When used in a skinning operation the relative weight of this bone on a point depends on where the point lies inside these capsules. If it is inside the inner capsule the relative weight is 1.0. If it is inside the outer capsule but not the inner one then the relative weight varies linearly between 0.0 at the outer capsule wall to 1.0 at the inner capsule wall. Outside the outer capsule the weight is 0.0.

The final weight on a point is found by dividing the relative weight by the sum of all the relative weights for all bones. If this sum is 0.0 then a default weight of 1.0 is given to the parent bone of the point.

Sequence:

=====

<sequence> :: = sequence { <\$string> [<figure-name>] <frames> ...
... <jointangle>* }

The name of the sequence is given by <\$string>.

<figure-name> :: = figure { <\$string> }

The name of the figure that this sequence applies to is defined by <\$string>.

<frames> :: = frames { <\$int> }

This chunk is the length of the sequence in frames.

<jointangle> :: = jointangle { <\$string> <\$string> <floattrack> }

The two strings give the names of the bone that it applies to, and the axis. The axis must be "X", "Y", or "Z".

Tracks:

<floattrack> :: = track { <floatkey>+ }

<floatkey> :: = floatkey { <\$int> <\$float> <\$string> }

<\$int> is the frame number for this key.

<\$float> is the value.

<\$string> is a string used to modify the behavior of this knot. It is currently ignored.

<pointtrack> :: = track { <pointkey>+ }

<pointkey> :: = pointkey { <\$int> <point> <point> <point> <\$string> }

<\$int> is the frame number for this key.

The first <point> is the value, the second <point> is the forward direction vector and this know, and the third <point> is the reverse direction vector.

<\$string> is a string used to modify the behavior of this knot. It is currently ignored.

<qtrack> :: = track { <qkey>+ }

<qkey> ::= qkey { <\$int> <quaternion> <\$string> }

<\$int> is the frame number for this key.

<quaternion> is the value.

<\$string> is a string used to modify the behavior of this knot. It is currently ignored.

<booleantrack> ::= track { <booleankey>+ }

<booleankey> ::= booleankey { <\$int> <\$int> <\$string> }

The first <\$int> is the frame number for this key. The second <\$int> is the value. It must be 0 or 1. 0 means False and 1 means True.

<\$string> is a string used to modify the behavior of this knot. It is currently ignored.

Scene:

=====

<scene> ::= scene { <\$string> <frames> <groundgrid> <shadowbias> ..

... [<background>] [<image>] [<fog>] ...

... [<znear> <zfar>] <element>* }

The name of the scene is in <\$string>.

<frames> ::= frames { <\$int> }

The number of frames. Must be greater than zero.

<groundgrid> ::= groundgrid { <\$int> }

If the value of <\$int> is one the ground grid is drawn. If it is zero then it isn't. <shadowbias> ::= shadowbias { <\$float> } The value used for shadow bias when drawing shadows. The nominal value is 0.001.

<background> ::= background { <\$int> <\$int> <\$int> }

The color used for the background. The three values are the red, green, and blue color values. 255 represents full brightness and 0 represents black. If not present the default color of 102, 102, 153 is used.

<image> ::= image { <\$string> [<panorama>] }

The name of a file used for the background image. If the panorama chunk is present then it is a panorama, otherwise it is a fixed background.

**<panorama> ::= panorama { <float-const> <float-const> <float-const> ...
... <float-const> }**

The four values are the left and right longitude angles and the bottom and top latitude angles of

the image. Looking into the screen in the front view is directly at <0, 0>. <-90, 0> is to the left, <+90, 0> the right.

<fog> :: = fog { <color> <fogstart> <fogend> [<fogpercent>] [<radial>] }

The <color> chunk is the color of the fog in 3 ints, from 0 to 255. <fogstart> and <fogend> are float chunks giving the starting and ending distance from the camera of the fog transition zone. <fogpercent> is an int chunk for the maximum fog level, if it is less than 100. <radial> is an empty chunk. If present then the fog distance is computed as the radial distance from the camera, otherwise it is the distance from the z = 0 plane.

<znear> :: = znear { <\$float-const>

<zfar> :: = zfar { <\$float-const>

If present, these set the distance of the near and far clip planes.

<element> :: = <camera> | <figureelement> | <objectelement> | <light> | <null>

These form the element of a scene. They share several common subchunks described here:

<\$string> - The first thing is the name of the element.

<loc> - A <point> for the element's location.

<orientation> - A <quaternion> chunk for the orientation, normally this is relative to its parent's orientation.

[<roll>] - An int chunk with a value of 1 if the element's orientation can "roll" off the vertical, and 0 if it can't.

[<facespath> | <facestarget> | <orienttarget>] - One of these three optional chunks may be present. They control whether the element's orientation is relative to 1) the direction of its path of motion, 2) the direction to another element, or 3) the same orientation as another element.

[<boundtarget>] - The name of the other element that this one is oriented to, if it has the facestarget or orienttarget chunk.

[<scale>] - A scale factor applied to the object before rendering.

<locked> - An int chunk with the value of 1 if this element is "locked" from being edited, and a value of 0 otherwise.

<controller>* - Zero or more <controller> chunks.

<element>* - Zero or more child elements, whose movement is relative to this element's coordinate system.

<camera> :: = camera { <\$string> <loc> <orientation> [<roll>] ...


```

... [ <facespath> | <facestarget> | <orienttarget> ] ...
... [ <boundtarget> ] [ <fov> [ <scale> ] ...
... <locked> <controller>* <element>* }

```

<fov> :: = fov { <\$float> }

The field of view in degrees of the camera's view from left to right.

```

<light> :: = light { <$string> <loc> <orientation> [ <roll> ]
... [ <facespath> | <facestarget> | <orienttarget> ] ...
... [ <boundtarget> ] [ <scale> ] <color> ...
... [ <infinite> | <local> | <spotlight> ] ...
... [ <castshadow> ] [ <percentshadow> ] ...
... [ raytraceshadow> ] [ <soft> ] ...
... <locked> <controller>* <element>* }

```

<color> :: = color { <point> }

The three values in <point> represent the red, green and blue components of the light's color. A value of 1.0 is maximum and 0.0 is minimum.

<infinite> :: = infinite { }

<local> :: = local { }

<spotlight> :: = spotlight { }

One of these three empty chunks must be present. They define the kind of light.

<inradius> :: = inradius { <\$float> }

<outradius> :: = outradius { <\$float> }

Spotlights and local lights can have a different intensity depending on the distance from an object. By default the intensity is the same. If these chunks are present then the intensity drops off linearly from full intensity at a distance of <inradius> to zero at a distance of <outradius>.

<inangle> :: = inangle { <\$float> }

<outangle> :: = outangle { <\$float> }

Spotlights also have a cone of influence. Within an angle of <inangle> to the direction of the light the intensity is full value. Between <inangle> and <outangle> it drops off linearly to zero. Outside this range it is zero.

<castshadow> :: = castshadow { }

If present this light casts a shadow.

<raytraceshadow> :: = raytraceshadow { }

This chunk enables ray tracing of shadows for this light.

<soft> :: = soft { }

If <soft> is present and the light casts a ray traced shadow the shadow is rendered with soft edges.

<softsize> :: = softsize { <\$float> }

The size of the soft shadow. For infinite lights this is the apparent width of the light source in degrees. For spotlights and local lights it represents the apparent diameter of the light as viewed by an illuminated surface.

<minsamples> :: = minsamples { <\$int> }

<maxsamples> :: = maxsamples { <\$int> }

Set the minimum and maximum number of samples used to calculate a soft shadow. An adaptive algorithm is used to speed up rendering when not in a "soft" region of the shadow.

<montecarlo> :: = montecarlo { }

Use true Monte Carlo sampling instead of pseudo-Monte Carlo sampling. This results in a grainier look in the shadows transition region.

<objectelement> :: = objectelement { <\$string> <\$string> <loc> ...
... <orientation> [<roll>]
... [<facespath> | <facestarget> | <orienttarget>] ...
... [<boundtarget>] [<scale>] [<visibility>] ...
... [<castshadow>] [<receiveshadow>] ...
... <locked> <controller>* <element>* }

Objects that are in a scene are defined by an <objectelement> chunk. The first <\$string> is the name of this element and the second is the name of the object that it references.

<visibility> :: = visibility { <\$int> }

This chunk has a value of 1 if the element is visible, and a value of zero if it is hidden. The default value is 1.

<castshadow> :: = castshadow { } If present this object casts a shadow.

<receiveshadow> :: = receiveshadow { }

If present this object shows shadows on its surface from lights that cast them.

<figureelement> :: = figureelement { <\$string> <\$string> <loc> ...
... <orientation> [<roll>]
... [<facespath> | <facestarget> | <orienttarget>] ...

```

... [ <boundtarget> ] [ <scale> ] [ <visibility> ] ...
... [ <castshadow> ] [ <receiveshadow> ] ...
... <locked> <namedsequence>* <controller>* <element>* }

```

Figures that are in a scene are defined by a <figureelement> chunk. The first <\$string> is the name of this element and the second is the name of the figure that it references.

```

<namedsequence> :: = namedsequence { <$string> <$int> }

```

This chunk specifies the name and starting frame of a sequence that is applied to this figure.

```

<null> :: = null { <$string> <loc> <orientation> [ <roll> ]
... [ <facespath> | <facestarget> | <orienttarget> ] ...
... [ <boundtarget> ] [ <scale> ] ...
... <locked> <controller>* <element>* }

```

A <null> chunk, also referred to as a "target" element in Anim8or, defines a new coordinate system. They can contain child elements and their location and orientation is fully animatable. They do not render in camera views.

Misc. Items:

```

<int-const> :: = [ + - ] <$int>

```

```

<float-const> :: = [ + - ] <$float>

```

```

:: = <int-const>

```

<point> :: = (<float-const> <float-const> <float-const>) A <point> is three possibly signed floating point or integer numbers enclosed in parenthesis. These normally represent a point in space or a direction vector but can also represent a color or any value made from three floating point values. There are no comma separators.

```

<quaternion> :: = ( <float-const> <float-const> <float-const> <float-const> )

```

A <quaternion> is four possibly signed floating point or integer constants enclosed in parenthesis. There are no comma separators. Quaternions represent rotations or orientations. They should be normalized (so that the sum of the squares of the four values is equal to 1.0) to insure correct behavior in Anim8or.

```

<name> :: = name { <$string> }

```

A <name> is chunk containing a string. Its value is used as a name within a particular chunk. There is normally no limit to what the string can be.

----- End of File -----

Appendix D

ASL Specification

Introduction

Anim8or introduces a scripting language ASL with version 0.95. It is included as part of Anim8or. A scripting language allows you to automate tasks that you do often, to extend Anim8or's basic functionality with plug-ins, and to animate elements in your scenes more flexibly and quickly.

ASL is an interpreted language loosely based on the C programming language but with a somewhat restricted set of functionality. It also borrows a few simple syntactic constructs from C++. Aspects of ASL that are the same as or similar to those in C use the same syntax. Those that are different generally use unique syntax to reduce the confusion that users familiar with C might have. Finally ASL includes a number of concepts specific to supporting Anim8or.

ASL scripts can be included within an Anim8or project or kept in a source file on disk. Normally disk files will have an extension of .a8s or .txt but any other extension may be used. The .a8s extension is specifically for scripts that will be preloaded when Anim8or starts up. The .txt extension is for scripts that are run from disk when using Anim8or.

This spec is for Anim8or v0.98 to Anim8or v1.0 . Additions to the scripting language in v0.98 over those in v0.95 are listed throughout the spec in **BLUE**. Additions and changes for the development version over those in v0.98 are shown in **GREEN**. Changes for v1.0 are listed in **RED** and those above v1.0 are shown in **Purple**.

Kinds of Scripts

ASL has several different types of scripts for various purposes. For example there are scripts that export Objects, those that build and manipulate models, and ones to compute the value of a controller in a Scene. They all use a common language but each has its own requirements and restrictions.

Variables and Names

Built-in variable and function names begin with a letter or an underscore character '_' followed by any number of letters, digits and underscores. Upper case letters are distinct from lower case so that 'Anim8or' and 'anim8or' are unique names. User declared variables must be preceded by a dollar sign '\$' as in '\$Anim8or'.

Data Types

The types supported by ASL are:

float

A **float** is a 32 bit IEEE floating point value. A float constant is a number with either a decimal point or an exponent or both:

1.234 .7 10. 1e10 3.141e-4

int

An **int** is a 32-bit signed integer. Constants may be decimal or hexadecimal:

1234 1 0x1234 0xface

void

The **void** type is used when defining functions that don't return a value.

```
void $print2(int $arga, int $argb)
{
    $file.print("$arga = %8.4g; $argb = %8.4g\n", $arga, $argb);
}
```

string

A **string** consists of zero or more ASCII characters. Constants are enclosed with double quotes. The backslash character `\` is used as a quote when double quotes are required in a string and to define non-printable characters. They are also used to define new-line characters and tabs as in C. Backslashes within constants must be doubled as well:

```
"hello"
"This is a string. "
"a double quote \" in a string"
"Line 1.\nLine 2.\n"
```

The string type has several useful functions.

```
typedef string {
    int length(void);
    string SubString(int first, int last);
    string Insert(string fStr, int pos);
    int GetChar(int pos);
    string SetChar(int val, int pos);
    string GetDir(void);
    string GetRoot(void);
    string GetExt(void);
};
```

The function `length()` returns the number of characters in a string.

`SubString()` returns a string composed of the first-*th* to the last-*th* characters in a string. Character positions are numbered from 0 to `length() - 1`. Out of bounds values are clamped to the dimensions of the string; if first is greater than last an empty string is returned.

`Insert()` inserts another string into a copy of a string before the character at position `pos` and returns the result. If `pos` is `<= 0` the new string is inserted at the first and if it is `>= length()` then it is added at the end.

`GetChar()` returns the value of the character at position `pos`. If `pos` is negative or `>= length()` it returns -1. `SetChar()` makes a copy of a string and sets the value of the character at position `pos` in the copy to `val`. If `pos` is out of range it does not change the string and returns the copy unaltered.

`GetDir()`, `GetRoot()` and `GetExt()` return the string corresponding to the director, root and extension of a string, respectively, assuming that it is a valid name of a file.

point2

The type **point2** is a vector of two floating point values. They are similar to the C struct:

```
typedef struct {
    float x, y;
} point2;
```

Individual floating point values are referenced using member notation:

```
$myvar.x    $myvar.y
```

Values are defined as a parenthesized list of two integer or floating point expressions:

```
(1.0, 7)    (0, 0)    (3.142, 1.732)    ($ii, 7*$p)
```

point3

Similarly the type **point3** is a vector of three floating point values. They are similar to the C struct:

```
typedef struct {  
    float x, y, z;  
} point3;
```

Individual floating point values are referenced using member notation:

```
$myvar.x    $myvar.y    $myvar.z
```

Values are defined as a parenthesized list of three integer or floating point expressions:

```
(1.0, 7, 9.999)    (0, 0, 0)    (3.142, 1.732, -1.414)  
($loc.x, $loc.y*1.5, 0.0)
```

quaternion

The type **quaternion** is a vector of four floating point values. They are used to represents a rotation or orientation in Anim8or.

```
typedef struct {  
    float x, y, z, w;  
} quaternion;
```

Individual floating point values are referenced using member notation:

```
$q0.x    $q0.y    $q0.z    $q0.w
```

Quaternion values may be constructed from 4 scalar values:

```
(1.0, 7, 9.999, -1e8)    (sin($angle), cos($angle), 0, 0)
```

float4x4

The type **float4x4** is a 4 by 4 element array. They are used for things like three dimensional transformations.

file

A **file** is a computer file for reading or writing data. They can only be used in specific kinds of scripts such as import and export plug-in scripts, and for debug output messages.

Various Anim8or Types

A number of types are used to refer to specific parts of an Anim8or project. Their internal structure is generally hidden. You can access and set various values in these types using built-in functions which are described later.

project - An Anim8or project.

object - An Object within an Anim8or project.

figure - A Figure in an Anim8or project.

sequence - A Sequence in an Anim8or project.

scene - A Scene in an anim8or project.

material - A material.

shape - A 3D component in an Object. These include Spheres, Cubes, Meshes, Subdivision

Objects, etc.

meshdata - The specific geometry of a shape as represented within Anim8or.

tridata - A simplified version of meshdata that is useful when exporting data.

arrays

Arrays in ASL are much more general than those in C. They are defined with an initial size but it can be increased or reduced by a running script. Elements of an array are referenced in the usual way:

```
$myarray[10]      $colors[$i + 12]
```

Additionally all arrays contain a struct like predefined member 'size':

```
$myarray.size
```

Referencing size returns the number of elements that the array currently holds. Assigning a value to size sets the number of elements to that value. When an array's size is increased in this manner new elements are assigned a value of 0, an empty string, or a null pointer depending on the type of the array. If the size is reduced the higher indexed values are discarded.

There are also two C++ like member functions for arrays pop() and push():

```
<type> $myarray.pop(void);  
int $myarray.push(<type> $value);
```

pop() removes the last elements from the array and returns its value, thus reducing the array's size by one. push() increases the size of an array by one and sets the value of the last element to its parameter. It returns the index of the element pushed.

Comments

You can use both C-style comment syntax. Text within comments is considered white space.

```
int $abc, /* $def;  
float      $ghi,  
           */ $jkl;      /* Variables $abc and $jkl are both declared as int. */  
int $varName;           // $varName is the name of an int variable.
```

Predefined Constants

There are several predefined constants in ASL. General constants are listed below. Other values that are only meaningful as parameters to certain functions are listed later with those functions.

```
int true = 1  
int false = 0  
float PI = 3.1415926;  
int VERSION;    // Current Anim8or version: v0.98x returns 98  
int VERSION;    // Current Anim8or version: v1.00 returns 100
```

Variable Declarations

Variables are declared similarly to how they are in C. However there are some differences:

- Variables cannot be initialized in the declaration.
- Declarations don't have to be grouped at the first. They can appear after executable statements, like in C++, but
- There are no scopes. A variable can be used anywhere after it is declared.

```
int $i, $count;
float $sizes[10]; /* array initially with 10 elements */
shape $mymodel;
float pie = 3.1415926; /* Not allowed */
```

Expressions

ASL supports many of the operators found in C.

Function Calls

ASL has a number of built in member and non-member functions. These are described elsewhere in this document. There is currently no support for user defined functions.

Unary Operators

```
-    negation: int, float, point2, point3, quaternion
!    not:      int, float
```

Binary Operators

```
+    addition: int, float, point2, point3
-    subtraction: int, float, point2, point3
+    concatenation: string
*    multiplication: int, float, point2/3*float, float*point2/3,
      quaternion*float float*quaternion
/    division: int, float
%    mod: int
left shift: int
>> signed right shift: int
< == <= comparisons: int, float, string;
> != >= returns int with value of 1 or 0
bit-wise AND: int
wise XOR: int
OR: int
&& logical and: int, float; returns int 1 if both operands are non-zero
|| logical or: int, float; returns int 1 if either operand is non-zero
++ -- prefix and postfix increment and decrement: int, float.
returns an expression, not an l-value.
```

Statements

Expression

An **expression** is the simplest statement. Normally an expression statement will call a function that has some kind of a side effect such as setting an Objects location. Expression statements that don't have side effects such as `$i + 1;` are allowed but aren't very useful.

Assignment

An **assignment** statement sets the value of a variable to value of an expression.

```
$index = 0;    $total = $part1 + $part2*1.5;
```

Numeric and string types simply copy the value of the expression to the variable:

```
int, float, string, point2, point3, quaternion, float4x4
```

Complex types refer to objects within Anim8or. They are actually handles to the objects, not the actual objects. Assigning a value to a variable of these types doesn't change the Anim8or project

but sets the object referenced by that variable. Member functions in these types are used to alter them.

material, meshdata, tridata, shape, object, figure, sequence,
scene, project

Variable of type file cannot be assigned.

Compound Statement

A **compound** statement is a list of zero or more statements enclosed in curly braces "{}". They may be used anywhere a statement may be used:

```
{  
    $shape.loc.x = sqrt($val);  
    $ii = $ii + 1;  
}
```

If Statement

An **if** statement evaluates a control expression. If the value is non-zero then the <then-statement> is executed next otherwise the <else-statement> is executed if it is present. The syntax is the same as C's:

```
if (<expr>)  
    <then-statement>  
else  
    <else-statement>
```

As in C the "else" statement is optional:

```
if (<expr>)  
    <then-statement>
```

While Statement

A **while** evaluates a control expression. If it the value of <expr> is non-zero then it executes its subordinate <statement> and reevaluates the control expression. This continues until the expression evaluates to zero. The syntax is the same as in C:

```
while (<expr>)  
    <statement>
```

Do While Statement

A **do while** statement begins by executing the body of the loop. It then evaluates a control expression. If the value of <expr> is non-zero then it executes its subordinate <statement> again. This continues until the expression evaluates to zero. The syntax is the same as in C:

```
do  
    <statement>  
while (<expr>)
```

For Statement

A **for** statement evaluates several control expressions and then executes its subordinate statement a number of times based on those values. For statements in ASL are not the same as those in C. The two general forms are:

```
for <var> = <init> to <limit> do  
    <statement>  
  
for <var> = <init> to <limit> step <step> do  
    <statement>
```

<var> can be either an int or float. The value of <init> is assigned to <var> and the values of

<limit> and <step> expressions are cast to the type of <var> and saved. If <step> is not present a value of 1 or 1.0 is used. Then the value of <var> is compared to <limit>. If <step> is greater than or equal to zero and <var> is less than or equal to <limit>, or if <step> is less than zero and <var> is greater than or equal to <limit> the <statement> is executed after which <step> is added to <var>. This procedure repeats until the comparison fails.

ASL For Statement

An alternate form of a **for** statement is supported to be compability with previous versions of Anim8or. It evaluates several control expressions and then executes its subordinate statement a number of times based on those values. These for statements do not have the same semantics as those in C. The <step> expression is optional:

```
for <var> = <init> to <limit> do
    <statement>
```

```
for <var> = <init> to <limit> step <step> do
    <statement>
```

<var> can be either an int or float. The value of <init> is assigned to <var> and the values of <limit> and <step> expressions are cast to the type of <var> and saved. If <step> is not present a value of 1 or 1.0 is used. Then the value of <var> is compared to <limit>. If <step> is greater than or equal to zero and <var> is less than or equal to <limit>, or if <step> is less than zero and <var> is greater than or equal to <limit> the <statement> is executed after which <step> is added to <var>. This procedure repeats until the comparison fails.

Break Statement

A **break** statement exits from the innermost enclosing for or while loop.

```
break;
```

Continue Statement

A **continue** statement finished executing the body of the innermost enclosing for or while loop and starts the next iteration.

```
continue;
```

Return Statement

A **return** statement exits from a user defined function. The type of <expr> must be assignment compatible with the type of the function. If the function returns **void** then there should be not return value. The syntax is the same as in C:

```
return <expr>
```

Special Type Details

The special ASL types are described here.

Project Type

```
struct project {
    object curObject;        // The current Object
    figure curFigure;        // The current Figure
    sequence curSequence;    // The current Sequence
    scene curScene;          // The current Scene
    int MarkMaterials(int val);
    int MarkObjects(int val);
    int MarkFigures(int val);
```

```

int MarkSequences(int val);
int MarkScenes(int val);
material NewMaterial(string name);
int GetNumMaterials(void);
texture NewTexture(string name);
texture NewCubeMapTexture(string name);
int GetNumTextures(void);
texture GetTexture(int index);
int GetTextureIndex(texture texVar);
texture LookupTexture(string name);
string GetDirectory(int index);
};

```

The Mark() functions set the mark member of all Materials, Objects, Figures, Sequences and Scenes, respectively, in a project to a value of 1 if val != 0, otherwise to 0. Mark() is useful for keeping track of whether a particular data item has been seen before when a script is exporting or altering a project. This is done by first clearing all marks with Mark(). The components of a project can then be visited in any order. If a component's mark member is zero it is set to 1 and the component is processed. If mark is already 1 that component can be skipped.

NewMaterial() adds a new global material to the project and returns it. If one already exists with the same name it doesn't add anything and returns NULL. GetNumMaterials() returns the number of global materials in a project. GetMaterial() returns a specific material. Materials are numbered from 0 to GetNumMaterials() - 1.

NewTexture() adds a new texture object to a project with the given name and returns the texture. NewCubeMapTexture() adds a new cube map texture. If a texture with that name already exists no new texture is created and NULL is returned. GetTexture() returns the texture numbered index, and GetTextureIndex returns the texture index corresponding to its parameter. If there is no such texture they return NULL and -1, respectively. LookupTexture() returns the texture with the given name or NULL if no such texture exists.

The function GetDirectory() returns the name of one of the target directories that Anim8or maintains. These are settable in the **File→Configure** dialog. The names shown below are used to access the different directories. An invalid value returns an empty string.

```

enum {
    DIRECTORY_PROJECT,
    DIRECTORY_TEXTURE,
    DIRECTORY_IMPORT,
    DIRECTORY_IMAGE,
    DIRECTORY_SCRIPT
};

```

GUI Type

```

struct GUI {
    int Xenabled;        // A value of 1 means the axis is
    int Yenabled;        // enabled and 0 disabled
    int Zenabled;
};

```

The GUI type is used only for a predefined variable GUI. It gives scripts access to certain settings in Anim8or's interface. Unless stated otherwise you can read and assign its members from within a script. Invalid or out of range values are silently clamped to a valid value or ignored.

Note: The GUI variable is not accessible from invariant scripts such as controller, parameteric plug-in or export plug-in scripts. Such scripts must return consistent results.

Material Type

```
struct material {
    string name;           // The current Object
    float Ka;              // Ambient factor
    float Kd;              // Diffuse factor
    float Ks;              // Specular factor
    float Ke;              // Emissive factor
    float alpha;           // Transparency 0->1, 1 = opaque
    float roughness;       // Specular roughness factor
    float brilliance;      // Brilliance factor
    int LockAmbientDiffuse;
    int Marked;            // 1 if Marked, 0 if not Marked
    point3 ambient;        // ambient color
    point3 diffuse;        // diffuse color
    point3 specular;       // specular color
    point3 emissive;       // emissive color
    texture GetTexture(int kind);
    int SetTexture(int kind, texture tex);
    int GetBlendMode(int kind);
    int SetBlendMode(int kind, int mode);
    int GetAlphaMode(int kind);
    int SetAlphaMode(int kind, int mode);
};
```

The `GetTexture()` function returns the texture for the specified kind if the material has one defined. Otherwise it returns `NULL`. Similarly the `SetTexture()` function sets the texture for the given kind. If `tex` is `NULL` then any currently defined texture of that kind is removed.

The following predefined names are accepted for texture kinds:

```
enum {
    TEXTURE_AMBIENT,
    TEXTURE_DIFFUSE,
    TEXTURE_SPECULAR,
    TEXTURE_EMISSIVE,
    TEXTURE_TRANSPARENCY,
    TEXTURE_BUMPMAP,
    TEXTURE_ENVIRONMENT,
};
```

These constants are integers and can be used in expressions like any integer constant. The actual values may change in future releases so be sure to use these names in your scripts instead of relying on specific integer values when calling `GetTextureName()`.

The functions `SetBlendMode()` and `GetBlendMode()` set and return a particular texture's blend mode. Similarly the functions `SetAlphaMode()` and `GetAlphaMode()` set and return a particular texture's alpha mode. The following enums correspond to the named modes.

```
enum BlendMode {
    BLEND_MODE_DECAL = 0,
    BLEND_MODE_DARKEN = 1,
    BLEND_MODE_LIGHTEN = 2,
};

enum AlphaMode {
    ALPHA_MODE_NONE = 0,
    ALPHA_MODE_LAYER = 1,
    ALPHA_MODE_FINAL = 2,
};
```

Texture Type

```
struct texture {
```

```

    string name;           // texture's name
    int Marked;            // Marked flag, 1 or 0
    const int CubeMap;     // 1 if a Cube Map, else 0
    int InvertImage;       // invert image flag, 1 or 0
    string GetFileName(void);
    string GetCubeMapFileName(int face);
    int SetFileName(string name);
    int SetCubeMapFileName(string name, int face);
};

```

Assigning to name changes the name of a texture and all references to it in a material. If another texture already has that name then the assignment is not made and the texture retains its existing name.

GetFileName() and SetFileName() return and set the name of the file used for the texture image. If the texture is a cube map then the functions GetCubeMapFileName() and SetCubeMapFileName() must be used instead. The additional parameter face specifies which faces value to use. Out of range values return null strings. The following predefined names can be used for the faces:

```

enum CubeMapFace {
    FACE_POS_X = 0,
    FACE_NEG_X = 1,
    FACE_POS_Y = 2,
    FACE_NEG_Y = 3,
    FACE_POS_Z = 4,
    FACE_NEG_Z = 5,
    FACE_RIGHT = 0,
    FACE_LEFT = 1,
    FACE_UP = 2,
    FACE_DOWN = 3,
    FACE_FRONT = 4,
    FACE_BEHIND = 5,
};

```

Attribute Type

```

struct attribute {
    string GetName(void);           // Return attributes name
    int GetType(void);              // Returns attributes type (from table below)
    int GetBoolValue(void);         // Value of a Boolean attribute
    int GetIntValue(void);          // Value of an integer attribute
    float GetFloatValue(void);      // Value of a floating point attribute
    point3 GetPoint3Value(void);    // Value of a point3 attribute
    quaternion GetQuaternionValue(void); // Value of a quaternion attribute
    string GetStringValue(void);    // Value of a string attribute
};

```

Calling a GetXXXValue() function for the a type other than the attributes actual type returns 0. You should call GetType() prior to querrying an attributes value to verify that it has the expected type. The following predefined names should be used for the types:

```

enum AttributeTypes {
    ATTR_UNKNOWN = 0,
    ATTR_INT = 1,
    ATTR_FLOAT = 2,
    ATTR_POINT2 = 3,
    ATTR_POINT3 = 4,
    ATTR_QUATERNION = 5,
    ATTR_STRING = 6,
};

```

Spline Type

```

struct spline {
    float GetLength(void);           // Return the length of the spline
    point3 Eval(float t);           // Value of spline at position t from the
start
    point3 GetTangent(float t); // Tangent (direction) of spline at t
    quaternion Orientation(float t); // Orientation of spline at t
};

```

You can use Eval() and Orientation() to sample the position and orientation of a spline. Use the function toFloat4x4() to convert the orientation into a transformation matrix.

Object Type

```

struct object {
    string name;           // object's name
    int Selected;          // Selected flag, 1 or 0
    int Marked;            // Marked flag, 1 or 0
    material NewMaterial(string name); // Define a new material
    int GetNumMaterials(void);
    material GetMaterial(int index);
    material LookupMaterial(string name);
    shape LookupShape(string name);
    int GetShapes(shape fshapes[]); // return array of top level shapes
    int GetNumAttributes(void);
    attribute GetAttribute(int index);
    attribute LookupAttribute(string name);
};

```

GetNumMaterials() returns the number of materials in an object and GetMaterial() returns the indexth material in the object. Materials are numbered from 0 to GetNumMaterials() - 1.

LookupMaterial() returns a material with the given name if there is one in the object, otherwise it returns NULL.

GetShapes() fills its parameter with all of the top level shapes in an Object. A single Group shape is returned for each top level group. Individual shapes in a Group can be accessed from this group with its GetShapes() member function. The return value is the number of shapes returned.

LookupShape() returns the shape with the given name if one exists. Otherwise it returns NULL.

Shape Type

```

struct shape {
    point3 loc;           // location relative to parent
    quaternion orientation; // relative orientation
    point3 bboxLo;        // read-only lower extent in local coordinates
    point3 bboxHi;        // read-only upper extent in local coordinates
    string name;          // shape's name
    int Selected;          // Selected flag, 1 or 0
    int Marked;            // Marked flag, 1 or 0
    int GetKind(void); // return kind of shape
    shape ConvertToMesh(void); // convert 3d shape to mesh
    shape ConvertToSubdivided(void);
    // convert 3d shape to subdivision shape
    float4x4 GetGlobalTransform(void); // Transform matrix
    float4x4 GetGlobalNormalTransform(void);
    // Transform matrix for normals
    meshdata GetMeshData(void); // return mesh information
    tridata GetTriangleData(void);
    // return triangular representation of shape
};

```

GetKind() returns the type of shape it represents. The values returned are:

```

enum {
    SHAPE_KIND_UNKNOWN,

```

```

    SHAPE_KIND_SPHERE,
    SHAPE_KIND_RECT_SOLID,
    SHAPE_KIND_MESH,
    SHAPE_KIND_CYLINDER,
    SHAPE_KIND_PATH,
    SHAPE_KIND_TEXT,
    SHAPE_KIND_MODIFIER,
    SHAPE_KIND_SUBDIVISION,
    SHAPE_KIND_IMAGE,
    SHAPE_KIND_PARAM_PLUGIN,
    SHAPE_KIND_GROUP,
    SHAPE_KIND_NAMED,
};

```

SHAPE_KIND_NAMED is used to refer to Objects from a Figure or a Scene.

GetMeshData() returns a structure with the description of the faces and materials defined by the shape. The meshdata structure holds the basic data used by Anim8or for a Mesh. Other shapes such as parametric shapes create meshdata structures for Anim8or when it needs to display them. Each point is uniquely stored in this format. Points used in multiple faces may use different normals or texture coordinates.

GetTriangleData() returns a tridata structure. This is a simplified representation of what is stored in a meshdata struct. It consists of triangles built from uniform arrays of geometry, texture coordinates, normals, etc. Points from a shape that are used in multiple faces with different normals or texture coordinates are duplicated. This representation is suitable for direct rendering as vertex arrays in OpenGL or D3D.

GetGlobalTransform() returns a matrix that transforms the points in a tridata variable to their location in the world view.

The following types are subclasses of the shape type. They have all of the members that shape has plus the additional ones shown. If you are unsure about it, you should check that a shape is the appropriate kind with GetKind() before accessing these specialized members as it will cause an error.

Cube Type

```

struct cube : shape {
    float xsize;        // x dimension
    float ysize;        // y dimension
    float zsize;        // z dimension
    int xdivisions;
    int ydivisions;
    int zdivisions;
};

```

Variable members like xsize can be both read and assigned. If assigned an out of range value it is silently clamped to the supported range.

Sphere Type

```

struct sphere : shape {
    float diameter;
    int lat;            // number of faces in latitude
    int lon;            // and longitude
};

```

Cylinder Type

```

struct cylinder : shape {
    float length;
};

```

```

float startdiameter;
float enddiameter;
int lat;           // number of faces in latitude
int lon;           // and longitude
int CapStart;      // 1 = cap start end, 0 = don't
int CapEnd;        // 1 = cap final end, 0 = don't
};

```

Mesh and Meshdata Types

The mesh and meshdata types are similar but not entirely equivalent. **mesh** is a subclass of shape which contains a mesh or subdivision shape. It is entirely editable and can have points, texture coordinates, materials etc. added, deleted or modified.

The **meshdata** type, on the other hand, is a read-only copy of an arbitrary 3D shape. It is used in scripts for exporting shapes and internally by Anim8or for drawing. Parametric shapes such as spheres create meshdata objects when Anim8or needs to draw them, for example.

There are some functions common to the two types that return internal values, and some functions found only in the mesh type that can modify it. Finally there are some functions found in both types that select and mark individual points, edges and faces that are useful for recording which parts have been processed but don't really change anything in them.

Mesh Members

The following functions in the mesh type are used to add, delete and change specific values:

```

struct mesh : shape {
    float smoothangle;
    int Open(void);           // Open a mesh for editing
    int Close(void);          // Finish editing a mesh
    int GetIsNew(void);       // True when a plug-in is first built

    // Adding points, texture coordinates, faces, etc.:

    int AddPoint(point3 loc);
    int AddTexCoord(point2 uv);
    int OpenFace(int matNo, int flags);
    int CloseFace(void);
    int VertexN(int index);
    int TexCoordN(int index);
    int AddMaterial(material mat);

    // Mesh altering functions:

    int SetPoint (int index, point3 val);
    int SetTexCoord(int index, point2 val);
    int SetMaterial(int index, material mat);

    // Point, edge and face value sets:

    int SetFacePointIndex(int faceIndex, int vtxIndex, int index);
    int SetFaceTexCoordIndex(int faceIndex, int vtxIndex, int index);
    int SetFaceMaterialIndex(int faceIndex, int matIndex);
    int SetEdgeSharpness(int edgeIndex, int val);

    int SetFaceHasTexCoords(int index, int val);

    // Delete functions:

    int DeletePoint(int pointIndex);
    int DeleteEdge(int edgeIndex);

```



```

    int DeleteFace(int faceIndex);
    int DeleteSelectedPoints(void);
    int DeleteSelectedEdges(void);
    int DeleteSelectedFaces(void);
    int DeleteMarkedPoints(void);
    int DeleteMarkedEdges(void);
    int DeleteMarkedFaces(void);

// Data clean-up:

    int RemoveUnusedTexCoords(void);
    int RemoveUnusedNormals(void);
    int RemoveUnusedBinormals(void);
    int RemoveUnusedMaterials(void);
    int RemoveUnusedData(void);
};

```

Meshes and subdivision shapes can be edited to add new points, faces, etc. A call to `Open()` enables editing. Once editing is complete `Close()` should be called to flush any unsynchronized data to the actual shape. These two functions return 1 if they succeed and 0 if they fail. Attempting to open a mesh that is already open or a shape that isn't a mesh or subdivision object will fail. Similarly calling `Close()` on an unopened mesh will fail.

`AddPoint()` and `AddTexCoord()` add new vertices and texture coordinates to a mesh. The return value is an integer index that is used to refer to the new point or texture coordinate when adding new faces. `AddMaterial()` adds a new material to the Mesh and returns its index.

`OpenFace()` begins the definition of a new face. It returns the index of the newly created face. If an error occurs it returns 0 and no new face is added. The parameter `matNo` sets the face's material. It is a zero based index into the mesh's materials. The bit vector flags specifies the additional properties the face will have. A value of `FACE_HAS_TEXCOORDS` specifies that the face will have texture coordinates. All other bits are ignored.

`GetIsNew()` returns true when a plug in mesh is first created, otherwise false. This allows a script to do one time initializations of materials and to preserve any material set by the user on subsequent calls.

`VertexN()` adds a new point to a face. `index` is the value returned by a call to `AddPoint()`. The return value is `index` if the call succeeds. If it fails the return value is -1. A call will fail if the value of `index` is not a valid point index for this mesh. `VertexN()` must be called after all other properties for a point have been set such as those set by a call to `TexCoordN()`.

`TexCoordN()` sets the texture coordinate for the current point. `index` is the value returned by a call to `AddTexCoord()`. The return value is `index` if the call succeeds. If it fails the return value is -1. A call will fail if the value of `index` is not a valid texture coordinate index for this mesh.

`CloseFace()` must be called after a face is defined to complete the definition.

`SetPoint()`, `SetTexCoord()` and `SetMaterial()` change the values of the point, texture coordinate or material indicated by the `index` parameter. This has the effect of moving a point, changing a texture coordinate, and changing a material for all faces in that mesh that currently use that value. The functions like `SetFacePointIndex()` however change the index number for a particular point in face number `faceIndex`. Other faces that used the same original point are not altered.

`SetEdgeSharpness()` sets the subdivision sharpness for an edge in the mesh. The edge will stay creased through this number of subdivisions. The sharpness can have a value of 0 to 7 where 0 is smooth and 7 is maximally creased.

SetFaceHasTexcoords() sets or clears the use of texture coordinates by this face. It returns the previous setting.

DeletePoint(), DeleteEdge() and DeleteFace() delete the point, edge and face corresponding to the value of the parameter. They return 1 if it was deleted, else 0 if it doesn't exist.

DeleteSelectedPoints/Edges/Faces() and DeleteMarkedPoints/Edges/Faces() delete and selected and marked component of that kind in a mesh. They return the number of things deleted. Note: after something is deleted the remaining components are renumbered. Thus if you have saved the index of a point in a variable it may no longer refer to the same point, or any valid point.

There are several functions that remove unused data from a mesh. They can reduce the amount of storage required for a mesh after its geometry has been edited. They don't delete any points or edges. What they do is remove unused auxiliary data that is not referenced by any geometry in the mesh such as texture coordinates or normals. RemoveUnusedTexCoords() removes unreferenced texture coordinates, for example. Note that RemoveUnusedMaterials() doesn't delete any materials from an object but it does remove unreferenced materials from a mesh's material table.

Note: Most mesh member functions cannot be applied to non-mesh shapes because the geometry of parametric shapes cannot be directly edited. However SetMaterial() can be applied to material index 0, the default material.

Mesh and Meshdata Members

The following functions in the mesh and meshdata types are used to access internal values:

```
struct mesh : shape { and struct meshdata {

// Metric queries:

    int GetNumPoints(void);
    int GetNumNormals(void);
    int GetNumTexCoords(void);
    int GetNumBinormals(void);
    int GetNumMaterials(void);
    int GetNumFaces(void);
    int GetNumEdges(void);

// Value queries:

    point3 GetPoint(int index);
    point3 GetNormal(int index);
    point2 GetTexCoord(int index);
    point3 GetBinormal(int index);
    material GetMaterial(int index);

// Face metric queries:

    int GetNumSides(int faceIndex);
    int GetFaceHasNormals(int faceIndex); add
    int GetFaceHasTexCoords(int faceIndex); add
    int GetFaceHasBinormals(int faceIndex); add

// Point, edge and face value queries:

    int GetFacePointIndex(int faceIndex, int vtxIndex);
    int GetFaceNormalIndex(int faceIndex, int vtxIndex);
    int GetFaceTexCoordIndex(int faceIndex, int vtxIndex);
    int GetFaceBinormalIndex(int faceIndex, int vtxIndex);
```

```

int GetFaceMaterialIndex(int faceIndex);
int GetEdgeIndex(int faceIndex, int index);
int GetEdgeSharpness(int faceIndex);
point3 GetEdgePoint0(int edgeIndex);
point3 GetEdgePoint1(int edgeIndex);
int GetEdgeIndex0(int edgeIndex);
int GetEdgeIndex1(int edgeIndex);
};

```

Mesh metric queries return the number of items of that kind in the mesh. The corresponding data value queries get the value of a particular one. `GetNumPoints()` returns the number of points, for example, and `GetPoint()` returns a specific point. Points, normals, binormals, texture coordinates and materials are zero based arrays. Thus the valid index for a point is 0 to `GetNumPoints()-1`.

Note: Edges are stored in a one based array however so valid indices for the edges in a mesh are 1 to `GetNumEdges()`. There is no edge number 0.

Face data queries return the parameters for a particular point or edge of a face. The first parameter is the face index and the second parameter is the point or edge index. The second parameter is an index into the 0 based array of points or edges and should have a value of 0 to `GetNumSides()-1`. `GetNumSides()` returns the number of sides or edges.

`GetNumEdges()` returns the number of edges in a mesh. Edges are numbered from 1 to N. `GetEdgeIndex()` returns the index of a particular edge in a face. The value returned can be positive or negative. If it is negative the edge is reversed when the face is viewed with the vertices in a clockwise orientation. `GetEdgePoint0()` and `GetEdgePoint1()` return the point for the beginning and ending points for an edge. `GetEdgeIndex0()` and `GetEdgeIndex1()` return the point index for the beginning and ending points for an edge.

The `GetFaceHas()` functions return true (1) or false (0) depending on whether the face has that particular property or not. For example `GetFaceHasNormals()` returns true if normal data is defined for the face.

The face value query functions return negative values when any of their arguments are out of range with the exception of `GetEdgeIndex` which returns 0.

Mesh and Meshdata Selection and Marking Members

The following functions in the mesh and meshdata types are used to mark and select individual points, edges and faces:

```

// Deselect or demark all points, edges or faces.
// Returns count of components cleared:

int DeselectPoints(void);
int DeselectEdges(void);
int DeselectFaces(void);
int DemarkPoints(void);
int DemarkEdges(void);
int DemarkFaces(void);

// Test individual points, edges or faces for mark or selection:

int GetPointSelected(int index);
int GetEdgeSelected(int index);
int GetFaceSelected(int index);
int GetPointMarked(int index);
int GetEdgeMarked(int index);
int GetFaceMarked(int index);

```

```
// Set or clear point, edge or face's selection or marking.
// Returns previous value:

int SetPointSelected(int index, int val);
int SetEdgeSelected(int index, int val);
int SetFaceSelected(int index, int val);
int SetPointMarked(int index, int val);
int SetEdgeMarked(int index, int val);
int SetFaceMarked(int index, int val);
}
```

Each individual point, edge and face may be marked and selected. Selected parts are often visible in the working views. The selected property is the same thing that is set when you select individual points, edges and faces when editing a mesh. The marked property is a temporary setting that is never visible. It is used throughout Anim8or to keep track of what parts of a mesh have been examined or altered during internal operations. Scripts can use these same properties when they examine any modify meshes as well. The marked property is transitory after a script has run. The selected property is preserved for mesh shapes but not for meshdata objects since meshdata objects are copies of the geometry of a shape, not the actual geometry.

The GetXXXSelected() and GetXXXMarked() functions test whether a particular point, edge or face in a mesh is selected and marked, respectively. The corresponding SetXXXSelected() and SetXXXMarked() set the respective selected and marked properties and return the original values. Finally the DeselectXXX() and DemarkXXX() functions clear all of the selections and markings for their respective kind of components.

Together these functions are useful when processing a mesh for export or modification.

Tridata Type

```
struct tridata {

// Tridata metric queries:

    int GetNumPoints(void);
    int GetNumTriangles(void);
    int GetNumMaterials(void);

// Tridata value queries:

    point3 GetPoint(int index);
    point3 GetNormal(int index);
    point2 GetTexCoord(int index);
    int GetIndex(int index);
    int GetMatIndex(int index);
    material GetMaterial(int index);
};
```

The tridata type is basically a set of arrays of data representing the points in a mesh. GetNumPoints() returns the total number of unique points, normals and texture coordinates it has. GetNumTriangles() returns the number of triangles, and GetNumMaterials() the number of materials.

The data for a particular triangle is indicated by the result of calling GetIndex() with three consecutive values. The number of points is not necessarily three times the number of triangles since points can be used in multiple triangles. To find the data for triangle N call GetIndex with 3*N, 3*N+1 and 3*N+2 where 0 <= N < GetNumTriangles(). The return values are passed to GetPoint() to find the value of the three vertices. The following code shows reads the data for

triangle N:

```
tridata $tdata;
int $N, $index1, $index2, $index3;
point3 $p1, $p2, $p3;

$index1 = $tdata.GetIndex($N*3);
$index2 = $tdata.GetIndex($N*3 + 1);
$index3 = $tdata.GetIndex($N*3 + 2);
$p1 = $tdata.GetPoint($N*3);
$p2 = $tdata.GetPoint($N*3 + 1);
$p3 = $tdata.GetPoint($N*3 + 2);
/* The values of $p1, $p2 and $p3 define triangle $N */
```

Float4x4 Type

```
struct float4x4 {
private:
    float mat[4][4];
    // Transform matrix - not accessible to scripts.
public:
    point3 Project(point3 p0); // Transform point by mat
};
```

Predefined Variables

There are several predefined variables in ASL.

project project;

The current project is kept in the variable project.

GUI GUI;

The graphical user interface settings are accessible through the variable GUI in non-invariant scripts.

float time;

The current Scene time in seconds is stored in time. Normally there are 24 frames per second but you can change this in a Scene's Environment settings. time has a value of 0.0 when not used in a Scene script. You cannot set the value of time by assigning to it. It is read only.

int frame;

The current Scene frame number is stored in frame. frame has a value of 0 when not used in a Scene script. You cannot set the value of frame by assigning to it. It is read only.

string version;

The version of Anim8or is available in version.

Functions

User Functions

As of v1.0 ASL supports user functions. The format is simialr to C functions:

```
<type> name ( <type> ident, <type> ident, ...)
{
    <statements>
}
```

There are several restrictions that apply to ASL functions as compared to C functions:

1. No forward declarations. Function headers must be followed by the definition in the form of a compound statement "{...}".

2. Recursion is not allowed. Anim8or doesn't check for self-recursion yet (a last minute bug). If you try it be prepared for chaos.

3. Functions must be declared before they are called.

4. The "main" function is called "\$main". It is optional. When not present, any statements that aren't in a function are gathered into a default "main" function, no matter where they appear. Kind of whacky, I know, but this allows you to more easily add functions to existing scripts.

5. No array parameters. All parameters and function results are the predefined types: **int**, **string**, **float**, **point2**, **point3**, **quaternion**, **float4x4**, **shape**, **meshdata**, **spline**, **material**, **attribute**, **texture**, **object**, **figure**, **sequence**, **scene**, and **tridata**.

6. All parameters are passed by value. Remember: types that represent objects within Anim8or (shape, meshdata, material, etc.) are actually handles so any changes to the parameter within a function affect the same object as outside the function. Future plans: support for an **out** modifier to return values through parameters.

7. Function format parameter and local declarations are in a separate scope from global variables. Declarations within compound statements { } are still in the enclosing scope however. I hope to fix this soon as well but it could potentially break existing scripts. Let me know if you think this would be a big problem.

Here's a simple function that returns the absolute difference between two squared floats.

```
float $absDiffSquared(float $argA, float $argB)
{
    float $answer;

    $answer = $argA*$argA - $argB*$argB;
    if ($answer >= 0.0)
        return $answer;
    return -$answer;
}
```

System Functions

System functions are available for all script types. They return various parameters about the current state of the active project.

```
void refresh(void);           // Refresh the screen
void view(string viewName);   // Change view
void render(void);           // Render an image
void renderer(string rendererName); // Set the renderer
```

After a script has run the screen is automatically updated to the current status. You can redraw the screen while a script is running to show its progress with the refresh() function.

You can change the view to any standard view with the view() function. The value of viewName must be on of the standard Anim8or views:

```
"front", "left", "right", "back", "top", "bottom", "ortho", "user1",
"user2", "user3", "user4", "perspective", "camera", "stereo", "all", "one"
```

all" changes to four view mode, and "one" changes to a single window of the focus view.

render() renders the current focus view with the currently active renderer.

renderer() sets the renderer. The value of renderName must match one of the available renderers. Normally this will include the following:

"scanline", "OpenGL", "ART Ray Tracer"

The available list for a particular computer is shown in the **Render→Renderer** dialog.

Math Functions

The following integer functions are supported:

```
int abs(int val);           // absolute value
int min(int a, int b);      // minimum
int max(int a, int b);      // maximum
int clamp(int val, int min, int max);
    // clamp val to min <= val <= max
```

The following floating point functions are supported:

```
float abs(float val);       // absolute value
float min(float a, float b); // minimum
float max(float a, float b); // maximum
float clamp(float val, float min, float max);
    // clamp val to min <= val <= max

float floor(float val);      // floor function
float ceil(float val);       // ceiling function
float fract(float val);      // val - floor(val)

float cos(float val);        // cosine
float sin(float val);        // sine
float log(float val);        // natural logarithm
float exp(float val);        // exp
float asin(float val);       // arc sine
float acos(float val);       // arc cosine
float sqrt(float val);       // square root
float tan(float val);        // tangent
float atan(float val);       // arc tangent
float log10(float val);      // logarithm base 10
float cosh(float val);       // hyperbolic cosine
float sinh(float val);       // hyperbolic sine
float tanh(float val);       // hyperbolic tangent

float pow(float val, float pow); // val raised to power pow
float atan(float a, float b);    // arc tangent of (a/b)

float lrp(float val, float a, float b);
    // linear interpolate between a and b:
    //     if (val < 0.0) return a;
    //     else if (val > 1.0) return b;
    //     else return a*(1.0 - val) + b*val;
```

The following vector functions are supported:

```
float length(point2 val);    // length of vector
float length(point3 val);    // length of vector
float length(quaternion val); // length of quaternion
point2 normalize(point2 val); // convert to unit length
point3 normalize(point3 val); // convert to unit length
quaternion normalize(quaternion val); // to unit length
float dot(point3 a, point3 b); // dot product of a and b
point3 cross(point3 a, point3 b); // cross product of a, b
```

The following special functions are supported:

```
point4 RPYtoQuaternion(float roll, float pitch, float yaw)
```

```

        // Compute the primary unit quaternion defined by
        // applying a roll , then a pitch, and finally a yaw
        // specified in degrees.
float4x4 toFloat4x4(quaternion val); // Convert a quaternion
        // into a transformation matrix

```

The following functions support pseudo random number sequences. The intent is to allow some variability but in a controlled manner for scripts that should be repeatable such as parameteric plug-ins and controllers. Thus the seed is always initialized to the same value at the start of a script. If you want to change the sequence for a parameteric plug in, set a new seed derived from a parameter. This way the script will produce consistent results.

```

int irand(void);           // 16b random value 0 to 65535
float frand(void);         // float random value -1.0 to 1.0
int randseed(int);        // set new seed and return current one

```

File Functions

The file type has several member functions. Files are currently only supported in Export Plug-In scripts, **Object Command scripts** and **Untyped scripts**.

```
int file.open(string fname, string mode);
```

The open() function attempts to open a text file using the value of fname. Relative path names start from the current working directory. If mode has a value of "r" the file is open for reading **a text file**, "rb" for **reading a binary file**, of "w" it is opened for writing, and of "a" it is opened for append which is writing at the end after anything already in the file.

Note: file input is only supported using Animor's built in scanner.

If the value of fname is "\$console" then the console (DOS text window) is opened for writing.

If the function succeeds it returns a positive value which represents the file handle, otherwise it returns zero.

It is an error to attempt to open a file that is already open.

```
int file.close(void);
```

The close() function closes an open file.

If the function succeeds it returns true (1), otherwise it returns false (0).

It is an error to attempt to close a file that is not open.

```
int file.IsOpen(void);
```

Returns 1 if the file is currently open, otherwise 0.

```
void file.print(string format, ...);
```

The print() function formats a string based on its parameters and outputs it to the file. It is similar to C's printf function but with a restricted set of formatting options. The formats supported are d, e, f, g and s. A maximum of 9 parameters are allowed after the format.

print() does not return a value.

```
string file.GetName(void);
```

The GetName() function returns the full path name of an open file. If the file isn't open it returns an empty string.

```
string file.GetDir(void);
```


The GetDir() function returns the directory of an open file. If the file isn't open it returns an empty string.

```
string file.GetRoot(void);
```

The GetRoot() function returns the root name of an open file. If the file isn't open it returns an empty string.

```
string file.GetExt(void);
```

The GetExt() function returns the extension of an open file. If the file isn't open it returns an empty string.

Text Scanner

The scanner type can be used to read txt files as a sequence of C-like tokens, automatically converting quoted strings and integer and floating point numbers into accessible forms.

```
struct scanner {
    int token;           // Current token kind.
    int intval;          // Integer value of current token if type is
ICONST_TOKEN.
    float floatval;      // Floating point value if token type is RCONST_TOKEN.
    string strval;       // String value for type SCONST_TOKEN.
    string ident;        // String value for type IDENT_TOKEN.
    int initialize(file <input-file>); // Initialize scanner. <input-file>
must be                                     // an "input" file. A "text" attributes
combines                                     // CR/LF and "binary" keeps them as
separate characters.                         // Returns the first token type int
file, and ERROR_TOKEN                       // on failure.
    int finalize(void); // Close scanner and free associated
memory.                                     // The associated file is left open.
    int scan(void);     // Scan the next token. Returns new
token type.
    void EOFisToken(int <flag>); // If <flag> is 0 then EOF is treated
as white                                     // space (default setting).
                                           // If <flag> is 1 then EOF returns
EOF_TOKEN.
    void scanToEOL(void); // Scan up to EOL or EOF. Scanned
characters are                                     // available in member strval.
    void DollarIsIdent(int <flag>) // Treat dollar sign $ as character in
identifiers.
};
```

Usage:

```
scanner $scan;
file $in;

$in.open($filename, "r"); // File can optionally be opened using a #file
directive.
$scan.initialize($in);    // Initialize with open input file, binary or text.
                           // First token is pre scanned.
$scan.EOFisToken(1);      // Optional. 1 returns EOL as a token. 0 skips
EOLs.
$scan.DollarIsIdent(1);   // Optional. 1 causes scanner to consider '$' a
letter
```

```

//          when scanning identifiers, 0 to consider it
a symbol.
do {
    if ($scan.token == TOKEN_ICONST) {
        $ivar = $scan.token;
    } else ...           // Process token stream
    $scan.scan();         // Scan the next token.
} until ($scan.token == TOKEN_EOF);
$scan.finalize();        // Free scanner memory. Does not close file.

```

Token values:

```

enum {
    ERROR_TOKEN,           // the scanner encountered an error
    EOF_TOKEN,             // end of file
    EOL_TOKEN,             // end of line
    IDENT_TOKEN,           // C-identifiers
    ICONST_TOKEN,          // Unsigned integer constant: decimal, hex or octal
format
    RCONST_TOKEN,          // Unsigned floating point constant
    SCONST_TOKEN,          // String constant.
    WCONST_TOKEN,          // L"xxx" - wide character string. Not currently
supported by ASL.
    LPAREN_TOKEN,          // (
    RPAREN_TOKEN,          // )
    LBRACE_TOKEN,          // {
    RBRACE_TOKEN,          // }
    LSQUARE_TOKEN,         // [
    RSQUARE_TOKEN,         // ]
    SEMI_TOKEN,            // ;
    COMMA_TOKEN,           // ,
    COLON_TOKEN,           // :
    HASH_TOKEN,            // #
    DOT_TOKEN,             // .
    LT_TOKEN,              // <
    GT_TOKEN,              // >
    LE_TOKEN,              // <=
    GE_TOKEN,              // >=
    EQ_TOKEN,              // =
    NE_TOKEN,              // !=
    EQEQ_TOKEN,            // ==
    AND_TOKEN,             // &
    XOR_TOKEN,             // ^
    OR_TOKEN,              // |
    ANDAND_TOKEN,          // &&
    OROR_TOKEN,            // ||
    EXCLAIM_TOKEN,         // !
    PLUS_TOKEN,            // +
    MINUS_TOKEN,           // -
    STAR_TOKEN,            // *
    SLASH_TOKEN,           // /
    PERCENT_TOKEN,         // %
    TILDA_TOKEN,           // ~
    PLUSPLUS_TOKEN,        // ++
    MINUSMINUS_TOKEN,      // --
    LSHIFT_TOKEN,          // <<
    RSHIFT_TOKEN,          // >>
    BACKSLASH_TOKEN,       // \
    CONCAT_TOKEN,          // ||
    DOLLAR_TOKEN,          // $
}

```

Miscellaneous Functions

ASL scripts also include this function:

```
string PrintToString(string format, ...);
```

The `PrintToString()` function formats a string based on its parameters and returns it as a string variable. It is similar to C's `sprintf` function but with the same restricted set of formatting options as `printf`. The formats supported are `d`, `e`, `f`, `g` and `s`. A maximum of 9 parameters are allowed after the format.

Shape Creation Functions

ASL scripts can define new basic shapes with the following functions:

```
shape sphere(float diameter);
shape sphere(float diameter, int long);
shape sphere(float diameter, int long, int lat);
shape cube(float size);
shape cube(float size, int divisions);
shape cylinder(float diameter);
shape cylinder(float diameter, float length);
shape mesh(void);
```

The `cube()` functions create a `Cube` with the same dimensions and number of divisions on all sides. If you want to set individual sides to different values you can assign those members in the cube directly:

```
shape $myCube;
$myCube = cube(10.5, 6);
$myCube.xsize = 20;
$myCube.xdivisions = 10;
```

The `cylinder()` functions create a `Cylinder` with the same dimensions at both ends and with the default number of divisions. You can assign different values by assigning the members directly:

```
shape $ myCylinder;
$myCylinder = cylinder(10, 50);
$myCylinder.enddiameter = 5;
$myCylinder.lon = 25;
$myCylinder.CapStart = 0;
$myCylinder.CapEnd = 1;
```

The `mesh()` function creates an empty `Mesh`. You can then add points and faces as needed with its member functions. See the description of the mesh type above for details. For example the following code creates a new `Mesh` with a single square face:

```
shape $square;
int $vtx[4], $tex[4], $i;
$square = mesh(); // Create a new mesh
$square.Open(); // Open for editing
$tex[0] = $square.AddTexCoord((0.0, 0.0));
$tex[1] = $square.AddTexCoord((0.0, 1.0));
$tex[2] = $square.AddTexCoord((1.0, 1.0));
$tex[3] = $square.AddTexCoord((1.0, 0.0));
$vtx[0] = $square.AddPoint(( 0.0, 0.0, 0.0));
$vtx[1] = $square.AddPoint(( 0.0, 10.0, 0.0));
$vtx[2] = $square.AddPoint((10.0, 10.0, 0.0));
$vtx[3] = $square.AddPoint((10.0, 0.0, 0.0));
$square.OpenFace(0, 4); // Start a new face with tex cords
for $i = 0 to 3 do {
    $square.TexCoordN($tex[$i]);
    $square.VertexN($vtx[$i]);
}
```

```
$square.CloseFace();      // Finish face
$square.Close();          // Close mesh or data won't be saved
```

Script Kinds

ASL scripts are used for a variety of different tasks. Each task has different requirements and restrictions. Plug-in scripts, for example, use a file name passed in from Anim8or and need to use that name when they are running. ASL uses **directives** to tell Anim8or how to pass information to and from scripts and often to tell Anim8or what kind of script it is.

Directives

Directives are source lines that begin with the hash symbol #. They must appear before any other statements but can appear after comments. The first directive typically defines what kind of script is being run. Any following directives link parameters and data to the outside world. Directive specifics are described next.

#command Directive

A **command directive** is used for a script that can be run from the menu in a particular editor. They are added to the Scripts menu for easy access when Anim8or first starts. To do this they must have the file extension ".a8s" and reside in the "scripts" directory. The format of a command directive is:

```
#command("<editor>");
```

Currently only the Object editor supports command scripts so the argument to the directive must be "object" as shown in the following example:

```
#command("object");
```

#plugin Directive

A **plugin directive** defines one of several kinds of plug-ins. The general syntax is:

```
#plugin("<editor>", additional parameters);
```

The number of parameters needed depends on what kind of plug-in the script is. The alternates are:

```
#plugin("object", "mesh", "<title>");
```

This defines a Parameteric Mesh plug-in script. The <title> is used to identify the Mesh type to the user. It should be short and it is better if it is a single word such as "spring".

```
#plugin("object", "export", "<title>", "<extension>");
```

This plug-in variant is for an Object export script. Such scripts are added to the Object→Export menu and work like built in exports. The title is used to identify the export kind to the user. The <extension> is the file extension for the output file. Some examples are:

```
#plugin("object", "mesh", "spring");
#plugin("object", "export", "Wavefront", ".obj");
```

#parameter Directive

A **parameter directive** defines a value that the user can pass to the script from Anim8or. They are only allowed for Parameteric Mesh plug-ins. The syntax is:

```
#parameter("<name>", <type>, <default>, <min>, <max>, ...);
```

Anim8or uses <name> in dialogs to set the value. < type> can be int or float. The default, minimum and maximum allowed values for this parameter are next.

These can be followed by one or more optional values that specify how the parameter scales when the user uses the Scale or Non-uniform Scale tool in the Object editor. The allowed values are:

```
scale    - parameter scales with the Scale tool.
scale_x  - parameter scales in x axis with non-uniform scale
scale_y  - parameter scales in y axis with non-uniform scale
scale_z  - parameter scales in z axis with non-uniform scale
```

Some example parameter directives are:

```
#parameter("sides", int, 6, 3, 16);
#parameter("diameter", float, 10.0, 0.01, 99, scale);
#parameter("offset", float, 20.0, 0.0, 99, scale, scale_x);
```

The parameter "offset" is scaled by both the Scale tool and by the x-scaling of the Non-Uniform Scale tool.

#return Directive

A **return directive** tells Anim8or where to find the result of a script that returns a value. The format is:

```
#return(<variable-name>);
```

The user variable must be declared in the script. For Export scripts the type must be int and be assigned a value of 1 if the export is successful, otherwise zero. For Parameteric Mesh plug-ins the variable should of type shape and returns the model created by the script.

Return directives are only allowed in Parameteric Mesh and Export plug-in scripts.

#file Directive

A **file directive** associates a variable with a file passed from Anim8or. The syntax is:

```
#file(<variable-name>, "text");
```

The variable must be declared in the script and be of type file. Currently only text files are allowed.

```
#file($output, "text");
```

A **file directive** associates a variable with a file that Anim8or prompts the user for before the script runs. The syntax is:

```
#file(<file-var>, <options>);
```

The options are:

```
"text"      - Process file in 'text' mode. Combine CR/LF characters into '\n'
"binary"    - Process characters individually.
"input"     - Open the file for reading.
"extension:<.ext>:<description>" - Use <.ext> file extension
                                in file open dialog, and <description> as description.
"title:<heading>" - Use <heading> as the title for user dialog.
```

The variable <file-var> must be declared in the script and be of type file.

For **Object Export Plug-In** scripts the option "text" is required and the other options are prohibited.

```
#file($uservar, "text");
```

For **Object Command** and **Untyped Command** scripts "input" is required.

```
#file($uservar, "text", "input", "extension:.obj:Wavefront .obj file",  
    "title:Select a Wavefront .obj File");
```

#button Directive

A **button directive** defines an image that Anim8or uses on a toolbar button. The syntax is:

```
#button(<width>, <height>, <num-colors>, <data> ...);
```

<width> and <height> are the dimensions of the image. < num-colors> is the number of colors. This must be 2. < data> is a comma separated list of 32 bit decimal or hexadecimal integer constants that define the image. The data is ordered by row starting at the top of the image. Each new row starts a new value.

Images with 2 colors are bit-masks. They use one bit per pixel. Zeros represent the background color and ones the foreground color.

Here is an example bitmap directive for a 17 pixel wide, 25 pixel high bitmap:

```
#button(17, 25, 2,  
    0x00000fc6, 0x00007079, 0x00008009, 0x00010f89,  
    0x00013871, 0x000107e1, 0x0000c003, 0x0000300d,  
    0x00000ff9, 0x00007079, 0x00008009, 0x00010f89,  
    0x00013871, 0x000107e1, 0x0000c003, 0x0000300d,  
    0x00000ff9, 0x00007079, 0x00008009, 0x00010f89,  
    0x00013871, 0x000107e1, 0x0000c002, 0x0000300c,  
    0x00000ff0);
```

which defines this button:



Specifics of Script Kinds

ASL has five different kinds of scripts. They are described next.

General Script

A **general script** can be run from the Scripts→Run-Script-File menu command. They normally do things like align the selected shapes, add new materials, or simple add some new shapes. If you give them a file extension of .a8s and they reside in the Script directory Anim8or will parse them when it loads and show any errors in the command window.

General Script Directives

General scripts cannot use any directives.

Command Script

A **command script** is a general script that is specifically built to run in a particular editor. They can do the same things that a general script can but if they use the .a8s file extension and are in the Scripts directory they will be added to the Scripts menu for easy access.

Command Script Directives

Command scripts are identified by a #command directive at the start. They cannot use any other directives.

Here is an example script that adds a Sphere of diameter 20 to the current Object, sets the longitudinal and latitudinal divisions to 16 and the location to (100, 20, 0):

```
#command("object");  
shape $mySphere;  
$myShape = sphere(20);  
$myShape.lat = 16;  
$myShape.lon = 16;  
$myShape.loc = (100, 20, 0);
```

Parameteric Mesh Plug-in Script

A **parameteric Mesh plug-in script** adds a new parameteric shape to Anim8or. Shapes of that type behave exactly like built-in shapes such as sphere and cube. They have a button in the left-hand toolbar and can be scaled, moved and rotated like other shapes.

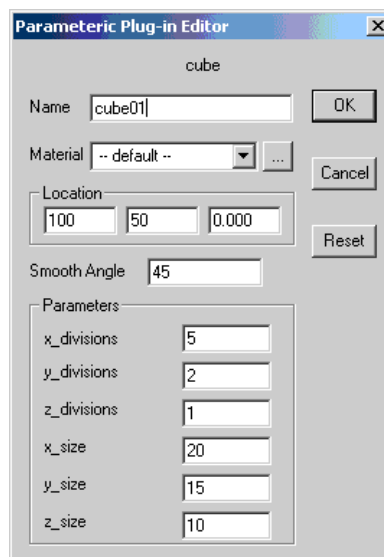
Parameteric Mesh Plug-In Script Directives

The first directive must be a mesh `#plugin` directive. There must be a `#return` directive for a variable of type shape. Normally there are one or more `#parameter` directives that define the parameters that the script uses to build the shape. Finally an optional `#button` directive defines the image for the toolbar button for this shape.

Here are the directives for a mesh plug-in for a cube:

```
#plugin("object", "mesh", "cube");  
#parameter("x_divisions", int, 1, 1, 100);  
#parameter("y_divisions", int, 1, 1, 100);  
#parameter("z_divisions", int, 1, 1, 100);  
#parameter("x_size", float, 10.0, 0.01, 1000, scale, scale_x);  
#parameter("y_size", float, 10.0, 0.01, 1000, scale, scale_y);  
#parameter("z_size", float, 10.0, 0.01, 1000, scale, scale_z);  
#return($cube);  
#button(24, 19, 2,  
    0x0000ffff, 0x00010843, 0x00021085, 0x00042109,  
    0x000ffff1, 0x00108431, 0x00210853, 0x00421095,  
    0x00ffff19, 0x00842111, 0x00842131, 0x00842152,  
    0x00842194, 0x00ffff18, 0x00842110, 0x00842120,  
    0x00842140, 0x00842180, 0x00ffff00);
```

When a user double clicks on a parameteric mesh in the Object editor a dialog is shown where the values of these parameters can be edited. These directives would make a dialog similar to this:



The current value of a parameter is accessed from a script with the parameter function. The type of the value returned by parameter() depends on the type in the #parameter directive:

```
float parameter(string name);  
int parameter(string name);
```

For example, the following code retrieves the values of x_size and x_divisions for the directives shown above:

```
float $xSize;  
int $xDiv;  
$xSize = parameter("x_size");  
$xDiv = parameter("x_divisions");
```

Parameteric mesh plug-in scripts cannot define any new shapes. Instead an empty Mesh is defined by Anim8or prior to running the script and assigned to the variable specified in the #return directive. The script adds the vertices and faces necessary to build the shape's geometry.

Object Export Plug-in Script

An **Object export plug-in script** exports an Object to a file in a new format.

Object Export Plug-In Script Directives

The first directive must be an Object export #plugin directive. There must be a #file directive linking the export file to the script, and an integer #return directive for the script to report success or failure when it finishes.

Here is an example of the directives for an Object export plug-in script:

```
#plugin("object", "export", "Wavefront", ".obj");  
#file($output, "text");  
#return($result);
```

Object export plug-in scripts can open additional files if needed.

Controller Expression Script

A Controller expression script computes the value of a controller in a scene. Any controller can use a script instead of a spline curve to compute its value. The only requirement is that there must be an assignment to a pre-declared variable named after the controller. In the simplest form a controller expression is just an assignment statement. Here is a script for an element's size controller:

```
$scale = 1 + sin(time*PI*4)*0.5;
```

The size of the element oscillates between 0.5 and 1.5 every 2 seconds.

Controller scripts can be more complex than a simple assignment. They can declare variables, use control flow, etc. They don't have a history however and cannot store data between frames.

Another useful property of Controller scripts is that they can refer to the value of other Controllers in Elements in the same Scene with the GetAttribute() functions:

```
int GetAttributeInt(string elName, string ctrlName);  
float GetAttributeFloat(string elName, string ctrlName);  
point3 GetAttributePoint3(string elName, string ctrlName);  
quaternion GetAttributeQuaternion(string elName, string ctrlName);
```

The type of the Controller must match the type in the name of the GetAttribute() function or a runtime error will be generated. The parameters must be string constants, not variables or computed values.

Here is an example that sets an Element's position to 50 units above the position of an Element with the name "Blue Element":

```
point3 $bluePos;  
$bluePos = GetAttributePoint3("Blue Element", "position");  
$position = $bluePos + (0, 50, 0);
```

Calling a GetAttribute() function in a script introduces a dependency on the order in which controller values must be computed. This is similar to what happens when one Element is set to face another element. You have to be careful to not create a circular dependency or Anim8or won't be able to decide what values to use. Anim8or will warn you when this happens, however, so it shouldn't be a problem.

Debugging

ASL scripts can be tricky to debug. More help is coming in this area so please be patient. Currently you can trace line numbers of statements as they are executed, and use print() to write selected data to an output file, but that's about it.

Tracing Script Execution

The Options→Debug→Trace-Script-Execution command toggles tracing the line numbers of each statement executed in scripts. When enabled the line number for each statement is output to the script window before it is executed. You can also enable or disable tracing in a running script with the debug() function:

```
int debug(int level);
```

Setting the level to 1 or higher enables tracing during the currently running script. Setting it to 0 disables it. The initial value for a script is always copied from the global setting. If tracing is enabled it will be 1, if disabled it will be 0.

Tracing can produce a lot of output and slow Anim8or considerably for complex scripts.

Appendix E

Learning ASL

Forum Notes

member: **polyGon_tError** (newbie)

webpage:<http://www.anim8or.com/smf/index.php/topic,4634.0.html>

If you want to learn ASL, have coding experience with any language this forum may help you to be with ASL.

ASL is based on C and is a interpreted Scripting language for Anim8or.

If you can programe in C language or understand coding in it, it will be helpfull.

If you know Action Script (flash), lingo (directory), python (blender, maya, & other)

C++ ,pascal ,visual basic ,java or even Java Script it will be helpfull too.

but there are few thing that may work different way :

1. ASL have no support for function, module, procedure, recursion, iteration, sub-routine.
2. ASL have no support for user input, event trace, interface access (have but limited).
3. ASL work top-down order way, no support for 'goto' like looping control.
4. ASL have 'for', 'if', 'while' support but check their usage in the manual.
5. ASL can't take all Array data in a single statement, from Linear-List/Data-Set like C.
6. you must use '\$' sign (like PHP) with all variable and other if needed.
7. ASL support complex, compound argument, parameter to default function.
8. ASL have no support for 2d, 3d Array, DataList and 'evalute' like function.
9. you can't use compound value assignment like $k+=1$ or $k++$, use $k=k+1$ for it.
 - a. ASL have no support for User-Defined-Data-Type, using many array is only solution.
 - b. ASL is for 3D mostly so a little knowledge of 3D (mostly OpenGL) will be helpfull.
 - c. you will be required to know about vector, matrix, trigonometry & physics as well to be able to make complex Script that can do amusing things!
 - d. ASL only support file Read / Write operation for Inport / Export Script.
 - e. timing based Scripting is supported in 'Scene' only with Object parameter.
 - f. ASL is limited to memory, very big array or long process can crash Anim8or.
 - g. there is a option (using while loop) to create inter-active / auto tool's.
 - h. debugging is possible only using Print statement with Consol, you can also output data / info to a file to analyse it manually.
 - i. you can't make edge directly, can move point, polyGon require to be replaced for modification, ASL can't feed selection data to a array like other do.
 - j. dealing with normal, texture co-ordinate can be tough some time.
 - k. if you use polyGon be aware of the vertex order, & winding way (usualy CCW).
 - l. some time using pre-defined data and logical pattern will be a good solution.
 - m. Set / Query operation for array are not present, can push / pop element to / from it of same kind.

- n.** use '.size' to know or set the size of array, set size to 0 to free up memory and element.
- o.** deleting anything like vertex, normal, edge, face re-index the rest of the them.
- p.** there is no way to keep any data in the memory for future reference (esp. in scene view).
- q.** using Console to feedback processing information to user can slow it down.
- r.** ASL support linear (mostly), compound (usually), complex (possible) way of logical processing.
- s.** ASL work in single script file, complex number is not supported.
- t.** ASL script have no undo support. esp. modeling / command script.

Before doing any thing check if it is done by another user, check the forum and other site for ASL or Anim8or Script (.a8s), if it is check if you can do it better or different way - to not to make anybody upset / unhappy.

You may use 'ASL Editor' (by Kubajzz) or 'Notepad2' (i use it with c++ scheme) to get help with bracket matching and easy / confuseless coding.

(note: You can also use the standard Notepad)

Best practice is to make clear what, where, how you want it to work, what will be needed, what you know for it, make a simple test check the output & your need. ask for help if needed (at least creating a temporary post for it, you will know).

only available source is :

webpage:http://www.anim8or.com/scripts/spec/Anim8or_Scripting_Language.html

(note: this is also found here in the current manual)

There were other source too, but now most are unavailable on the net.

best option is to search for '3D', 'OpenGL' and '3D math' related website / pdf.

basicaly all script are open source (if somebody ask for a credit give it!).

take what you want, do how you like it, show us - we will love to see it.

** usual Scripting process is listed here check to make sure you do it correct way.*

ASL usage in Anim8or are as follow :

Parametric-Object Script :

- > define all the variable / array (can be done on the other part too)
- > define parameter for the object
- > use logic and math with the data in process to build the object
- > build the object polyMesh from it
- > return it to Anim8or

Modeling Tool / Function Script :

- > define all the variable / array (can be done on the other part too)
- > take selected / working mesh, shape & other (can be multiple) in some array
- > get all mesh, mesh part (vertex, edge, face, normal, texture co-ordinate, material) data and isolate selection area data if needed
- > use attribute to get user optional data
- > use logic and math to analyse & process the data
- > apply it to the working mesh (change or re-define as needed)

Import Script :

- > define all the variable / array (can be done on the other part too)
- > open a file needed to import from
- > parse it as string using string function and logical process
- > define the object, use processed data to build the object

Export Script :

- > define all the variable / array (can be done on the other part too)
- > take selected / working mesh, shape & other (can be multiple) in some array
- > get all mesh, mesh part (vertex, edge, face, normal, texture co-ordinate, material) (selection or all object) required to export to a file format
- > use logic and math to analyse & process the data for the file format
- > open a file with extension as needed
- > export the data to the file

Controller Script :

- > select Object parameter to assign controller script
- > define all the variable / array for it
- > location, orientation, scale, visibility, other few parameter can be changed
- > bind them logically & using math with time to animate them
- > play the scene to test, render to get output

good to remember :

- * delit all array data at the end of their usage, esp. reseting size to 0.
- * delit all orphen data at the end of the script, esp. polymesh data like normal, texture coordinate.
- * comment big, complex script worthfully for future reference.
- * give variable meaningfull, related-to-subject like name.
- * give sort discription of script function, usage.
- * keep little information about ownself within script, avoid giving email address.
- * put all required data (selection, working) in array to speed up processing, instead of findind them every time needed.
- * new created object, vertex, edge, face like data are indexed at the end so check reverse way, check after recent to find, use them quickly if needed.
- * any thing like object, vertex, edge, face if deleted the rest is reindexed so to find privious one build an array with them, while deliting using original index value reindex them negeting number of time deleted, if possible put them in a array select / mark them then delit them at the end.
- * if creating inter-active / auto tool's leave an option to end it manually.
- * avoid creating more then 2 face sharing edge build polygon.
- * if normal, texture co-ordinate, material data not working correct way leave them as is.
- * avoid creating very big or very small face, subdivide or merge with coding if possible.
- * optimize and test script for big, regular, irreguler, patterned, ordered, seperated polymesh form.
- * make sure polygon have correct vertex order (and winding way) or it will be invalid (and flipped).
- * use logic to avoid unwanted modification by command script, undo is not possible.

« *Last Edit: January 07, 2014, 07:32:11 am by polyGon_tError* »

Anim8or Scripts

webpage:<http://www.anim8or.com/scripts/index.html>

(For access to these scripts you can find the links listed at the address above.)

R. Steven Glanville

Parameteric Shape Plug-Ins

spring_plugin_1.a8s -This plug-in makes a spring. You can scale the size, and the number of cycles and thickness of the wire with the non-uniform scaling tool. Other properties you can set in the shape's dialog include the number of sides used in the polygon and the spread between turns.

Object Export Plug-Ins

.obj -This plug-in exports an Object to the **.obj** format. It behaves very much like the one built in to Anim8or except that it also creates a **.mtl** file.

.c -This plug-in exports the same C language data structures that the built in exporter does. You can easily modify it for your own uses.

Sample Scripts

Here are some random script examples for you to examine and experiment with. They are sort of a mish-mash of things that the scripting language can do but I thought you might find them interesting until I finish the language spec and write some better ones. They are not plug-ins. However if you store them in the Scripts directory they will be easily accessible from the Scripts menu for you to run and edit.

Note: if you give them the ".a8s" extension and store them in the Scripts directory Anim8or will parse them when it starts running. They won't be installed as plug-ins but you will be able to see if they contain any errors by examining the command window output.

array_2.a8s -This script adds **Spheres** of various sizes and locations.

material_1.a8s -This script defines some new **materials**. You have to open the Material toolbar in order to see what it does.

for_3.a8s -Here's an example of how to **render** form a script. You can't save the results. They just flash by and disappear.

cube.txt -Define some **Cubes** and edit their parameters with this one.

cylinder_2.txt -Do the same for some **Cylinders** but also **convert** one into a **Mesh** and another into a **Subdivision Mesh**.

mesh_1.txt -**Build a Mesh** point by point, and face by face

sphere.txt -Adds a couple of Spheres, **refreshes** the screen, **changes the view**, and renders an image.

Scripts from Users

Here are some scripts from users:

plugicon.zip -A program to make **script icons**. **Author: Tyson Collins**

export_gm6_plugin.a8s -This popular script exports models to **Game Maker 6** format. Don't forget to copy it to your "scripts" directory to use it (like you have to do for all plug-in scripts). **Author: Tyson Collins**

export_cmod_plugin.a8s -Here's an export plug-in script that writes the **cmod** files for use with **Celestia**, a 3D astronomical visulation program. **Author: selden**

cone_plugin.a8s -Here's a **cone** parameteric shape. **Author: Tyson Collins**

torus_plugin.a8s -Here's a **torus** (also called a donut by all you Homer Simpsons fans out there). **Author: Tyson Collins**

plane_plugin.a8s -Here is a simple **horizontal plane**. If you want to study a simple plug-in script this is a good one to look at. **Author: Tyson Collins**

K_Hair.a8s -Kevaniz made this parameteric shape script to help make **hair**. You'll also need to put this texture: **K_Hair_Texture.gif** into your texture directory. Here's a link to both of these files zipped together with a tutorial video on how to use it (webpage:<http://www.freewebs.com/kevinaz/K%5FHair.zip>) **Author: Kevaniz**

teapot.a8s -Kubajzz wrote this parameteric shape script that models the iconic Utah Teapot. A real classic for computer graphics history enthusiasts! **Author: Kubajzz**

diamond.zip -Another great parameteric shape script by Kubajzz models 7 different cuts of **diamonds**. Render them using Anim8or's ART ray tracer for a realistic gem of an image! (Don't forget to read the ReadMe file and the Explanation.jpg to get the most out of these little jewels.) **Author: Kubajzz**

snailshell.a8s -Here's a great parameteric shape scripts for making **sea shells** and **snails**. **Author: NickE**

spline.txt

NickE has modified the spline.txt script to make **chains**. There are two versions:

chain_make.txt -is a command script. To use it you define a spline and select it. Then you run the script with the Scripts→RunScript command. **Author: NickE**

chain_maker.an8 -is a parameteric parameter script. Make a spline and name it "chainpath". Then click the chain icon in the left hand toolbar and click in the modeling window. (The .a8s file needs to be in your scripts directory.) Parameteric scripts aren't really supposed to work this way so you need to be careful and not rename the spline.**Author: NickE**

ASL Scripts Database

webpage:<http://www.anim8or.com/smf/index.php/topic,1705.0.html>

(For access to these scripts and some third party tools you can find the links listed at the forum topic address above.)

Located here you will find a listing of ASL scripts along with their descriptions, download, author, thumbnail/preview, and links.

Note: If you own one of these scripts and do not wish for it to be listed and hosted here then let me know and I will take it down.

So you want to submit a script? No problem! Just post a reply to this topic (listed above) with as much information as possible in the following areas:

1. Script Name
2. Author Name
3. Download link
4. Description
5. Related links (points to one or more web pages that explain what the script is and/or how to use it)
6. (Optional but helpful) If it's a shape plugin, supply a 30x30 gif of the button like how the listing has it, and if it's a command script supply up to a 52x52 animated gif with transparent background showing it in action.

What's listed so far isn't everything but I'm working on it. Hopefully this will become a very, very, very long list

Raxx

Shape Scripts

3D PI Chart

Create any sized chunk of cylinder similar to a pie based on Divisions, Radius, Texture Scale, Start Angle, End Angle, and Depth.

Author: BOB_I_Ts

4 Wall

Basic frame shape that allows you to create a square-shaped "hole in the wall". You can adjust the Horizontal Scale, Vertical Scale, Depth, and the hole's offsets and sizes.

Author: BOB_I_Ts

Advanced Plane

Creates a simple rectangular plane. You can change the width, height and the number of divisions.

Author: Vobla

Archway

Creates an archway block. You may define the width, height, length, divisions, ceiling depth, and angle.

Author: BOB_I_Ts

Cage

Create a variety of cage shapes. You may specify the XYZ Size and Divisions, the thickness of the wire, and if it needs to be a single wall or a full cubic cage.

Author: Kubajzz

Cog

Cog Wheel shape. Many different parameters make for a large variety of cog wheels available.

Author: BOB_I_Ts

Diamond

Make diamonds of any shape and size with this plugin. You can change the general shape along with the size, crown and pavilion height, X-Z ratio, Corner size, and Culet size.

Author: Kubajzz

E_Track

Instead of extruding splines along a spline path, this plugin allows you to extrude a mesh shape along a mesh path made out of points.

Author: Brian Heath (B_twist)

F8ce

An an8 version of MakeHuman, allowing you to easily create a variety of humanoid characters.

Author: BOB_I_Ts

Jet Engine Blades

A script that makes jet engine blades. Define the number of blades, height, width, center width, size, and inner & outer rotation.

Author: Monex

LShape

Lets you create an L-shaped block or corner block. You can customize the Horizontal and Vertical Length and Width of each leg and the depth.

Author: BOB_I_Ts

Paraboloid

Creates a paraboloid shape. Lets you specify the size, height, and vertical and horizontal segments.

Author: Kubajzz

N-Sided Polygon

Good replacement to the N-Gon tool if you're just looking to make an N-sided plane. Options for the number of sides and toggle a center division in the parameters.

Author: Kubajzz

Random Terrain

Generate a randomly shaped terrain.

Author: Felipe Manga

Ring shape

Simple and useful ring/pipe primitive.

Author: NickD

Roof (Triangular Prism)

Simple roof primitive in the form of a triangular prism. You may define the height, width, depth, and the offset of the peak.

Author: BOB_I_Ts

Slope

Create a simple slope with the option to customize the height, length, and starting/ending width and scale.

Author: BOB_I_Ts

Spring

Original spring shape by Steven Glanville. You may define the sides, diameter, offset, segments, cycles, and spread.

Author: Steve

Stair Shape

Easy way to make a staircase by defining the number of steps, length, height, and width. You can also specify if the stairs need a solid support or not.

Author: BOB_I_Ts

Star

Create a nice star . You can define the number of beams, radius, thickness and more parameters.

Author: *Syziph*

Utah Teapot

The classic 3D teapot.

Author: Kubajzz

Tetrahedron

Simple tetrahedron shape (3-sided pyramid).

Author: Kubajzz

Torus

Torus or donut, you call it!

Author: *Tyson Collins*

UV Block

Builds a cube with proper UV coordinates on all sides. If viewing from the front viewport you can adjust the UV scaling for all the sides when using the non-uniform scale tool.

Author: BOB_I_Ts

Command Scripts

Add Edge Loop

More than just creating an edge loop based on an edge you select, it will try to deform it based on the surrounding mesh (make it contoured).

Author: Kubajzz

Apply Selection

Select some points, store it, then apply that selection later on!

Author: Claude

Boolean Operations

Create a wide variety of shapes easily using this plugin! Enables union, intersection and subtraction.

Author: Kubajzz with the help of NickE

Complex Cleanup

This script will delete completely useless edges and points and it will select the points and edges you should edit to keep your mesh clean. If you want to make nice clean models, you should run this script everytime you finish a model (especially if you are a beginner...).

Author: Kubajzz

Connect Points

Connect two points of the same face and split the face in two (a knife tool)

Author: Kubajzz

Copy Faces

Do you need to duplicate a part of your mesh? You can use this tool. It does basically the same like the Detach Faces command, but the original shape is not changed.

Author: Kubajzz

Deform Mesh

Deform a mesh based on strength, diversity, smoothness, seed, and refresh attributes.

Author: Kubajzz

Edge Spin

Just select an edge that splits a 4+ sided polygon and spin it!

Author: Claude

Grow Point Selection

Select one or more points and this script will select the next nearest connected points to them.

Author: Kubajzz

Poly Reducer (Reduce Triangles)

Reduces a triangulated mesh anywhere between 1%-95% while preserving materials and UV Coords.

Author: Kubajzz

Reload Textures

Reloads all textures in your project.

Author: Kubajzz

Remove Edge Loop

Select an edge loop and this script will remove it, keeping the mesh intact.

Author: Kubajzz

Remove Useless

Removes useless points on an edge that don't serve any function. This usually occurs when merging faces.

Author: Kubajzz

Select Similar

Select one or more meshes, faces, edges, or points and this script will find similar structures and select them as well.

Author: Kubajzz

Snap2Pnt

Set a point as the snap-to point, then snap any number of other points to it!

Author: Claude

Triangulate Mesh

Creates a triangulated copy of a mesh.

Author: Claude

Two Point Distance

Finds the distance between 2 points.

Author: CobraSpectre

Universal Mirroring

Need to mirror several shapes at once? This plugin can do that. It can also mirror groups, subdivisions, parametric meshes, spheres and more...

Author: Kubajzz

Unsubdivision Script

Have you ever permanently converted a subdivision object to mesh and wanted to convert it back, or picked up a previously subdivided mesh from someone else? Reverse it using this tool!

Author: Kubajzz

Untriangulate Mesh

Untriangulates a triangulated mesh!

Author: NickE

Export Scripts

.C

Exports the same C language data structures that Anim8or's built-in exporter does. Does not export animations.

Author: Steve

Celestia (.cmod)

Format for a popular space simulation program. Does not export animations.

Author: Selden

Orbiter Mesh (.msh)

Exports to a format compatible with the Orbiter program

Author: Urwumpe

Roblox (.mesh)

Exports to the .mesh format compatible with a popular brick-based game. Does not export animations.

Author: Gamer3D

STL (.stl)

Exports the basic triangulated models into a regular .stl file. Does not export animations.

Author: Raxx

VRML (.wrl)

Exports objects to the Virtual Reality Modelling Language.

Author: Paulo

WaveFront Object (.obj & .mtl)

Updated to include more material properties and multiple materials per mesh. Does not export animations.

Author: Steve

Updated by: MrProtek

.X (by Raxx)

This is an updated .X exporter that is a mix of the best of the other .X exporters, and may be more compatible with your program of choice. Does not export animations.

Author: Raxx

.X (by Joe Cooning)

Exports to the popular DirectX/D3D format. Does not export animations.

Author: Joe Cooning

.X (by zaidon)

Based on Joe Cooning's script, this version exports to the popular DirectX/D3D format but imitates the Conv3DS output structure. Does not export animations.

Author: zaidon

.X (by Cranny76)

Exports to the popular DirectX/D3D format. Remarkd to be a rough draft version. Does not export animations.

Author: Cranny76

WRL (.x3d)

Exports to the .X3D file format.

Author: hermetic cab

Complete History of ASL forum Scripts

webpage:<http://www.anim8or.com/smf/index.php/topic,5634.0.html>

Complete history of scripts posted in ASL forum . (For access to these scripts you can find the links listed at the forum topic address above.)

Gyperboloid

Sr. Member

« on: October 01, 2017, 12:34:53 am »

Huaha :-X

Eyes bleeding

So. I returned to Anim8or after a long time not been around. Lot of changes and as I mentioned a couple of times, I lost the ball here. So I decided that if I want to catch up the whole thing I shall start all over again. From zero, totally , as I did never before. :P I searched all new videos, tutorials that were created from the point I kind of moved away a little bit . Lot of new stuff, but not that much. Can't say the same about the development of Anim8or . Huge steps forward. So I thought the only way (as goes with my way of thinking, behavior in general) is to go through the whole Anim8or forum !!! Started a little bit on some threads in all forums, but then decided to "clean up " the whole ASL forum for my self (as known, scripts are the power !!!) . So I came up with a list of scripts that were posted from The Big Bang the day of birth of the Official Anim8or Forum (the new one) . First goes the name of the topic in which originally were posted the scripts and finally the scripts of interest and the nickname of the author of the script. There are a few programs and some tools beside them also, just few. There are also some interesting semi-finished or just some wip scripts,controllers and stuff that never reached (at least officially) their release point, and other stuff that certainly may present some interest to those who are interested in scripting, programming e.t.c, but I didn't include them. Here you'll find only the officially released, finished tools , that were ever posted on forum. Starting from the oldest post.

new - advanced_plane_plugin - available - "Advanced Plane" script by <<vobla>>

new - line_points script - might be available, if you'll help me here - "Snap to point" script by <<Claude>>

Car script for fun - "car" script by <<animtime>>

script: 5 vehicles - "fighter","hearse","rallycar1","stealth", "warrior" scripts by <<animtime>>

Cage Parametric shape - "cage" script by <<Kubajzz>>

The Utah teapot - "The Utah teapot" script by <<Kubajzz>>

Object Script that Generates Scene Controller Scripts - "path to controller" script by <<NickE>>

Script : Copy "Source" mesh to " target " mesh - " mesh2mesh_6_wnfitt_final" script by <<NickE>>

Script Request : Chain Link Maker - " chain_maker_vls_1", "chain_maker_pm_vls", " spline_tube_1 " scripts by <<NickE>>

Diamond parametric shape released - "diamond" script by <<Kubajzz>>

Script to Generate Snail Shells - "snailshell" script by <<NickE>>

Find Distance Between Two Points Script - "Two_Point_Distance" script by <<CobraSpectrum>> edited by <<Kubajzz>>

EXPERIMENTAL Mirroring Script - "mirroring script" by <<CobraSpectrum>>

Old Magnet Tool - " automagnet" script by <<Francesco>>

Sage ASL collection - "metalics" script by <<sagedavis>>

Magnet Tools Package - "Magnet Tools 08-11-20" scripts by <<Francesco>>

Please help with the cylinder script! - "dome_5_8","dome_6_8","new-dome_5_8" scripts by <<NickD>>

Anim8or V 1.0 build 1.01.1318 Manual

Customizable 3D ship hull script (demo included) - "springsharp" script by <<NickD>>

PointSplit script testing - "PointSplitBeta1" script by <<Claude>>

HELP in Script (tree script) - "05_tree" script by <<Claude>>, edited version of <<neirao's>> "tree script"

"Ring" parametric shape plugin - "ring" script by <<NickD>>

script request-copy along a spline - "mesh2path_2" script by <<NickE>>

Centre hotpoint to centre of mesh script request - "center meshes on axis" script by <<NickE>>

Boolean operations plugin - "BooleanOperations" scripts by <<Kubajzz>>

Copy Faces plugin - "Copy Faces" script by <<Kubajzz>>

Universal mirroring plugin - "UniversalMirror" script by <<Kubajzz>>

VRML exporter plug-in - "export_vrml_plugin" script by <<Paulo>>

Script Request: untriangulate faces - "untriangulate_mesh_3" script by <<NickE>>

Materials Tutorials - finished - "tutorials" (not scripts) by <<Jdez>>

Request for a script - position of camera relevant visibility controller script by <<BOB_I_Ts>>

F8ce primitive - "f8cev7" script by << BOB_I_Ts>>

Controller Scripts: Motion simulation based on Physics' Kinematic Equations - "BouncyBall" an8 file with controller script inside it by <<hf003>> and "script_rotation_XYZ" an8 file with orientation controller script inside it by <<neirao>>

Ani2svg - "ani2svg" programm by <<dwsel>>

collapse points command - "collapse points " script by <<BOB_I_Ts>>

Simple fluid simulation - "ripple_delay" script , "xzrippletest_2" an8 file by <<NickE>>

"roofs" script by <<freesailor>>-setting materials in ASL

Converting a mesh to script - is this possible? - "Script Export" script by <<NickD>>

PointBevel script testing - "PointBevelBeta1.0" script by <<Claude>>

ALIGN POINTS SCRIPT - Help need - script by <<neirao>> edited by <<NickE>>

Ani2lux 0.1 alpha - "ani2lux 0.1.1a" programm by <<dwsel>>

Anim8ing a Tank Tread - "website_www.tranceportsoftware.com/anim8orstuff/treads.htm" sourcewith scripts needed on it by <<NickE>>

Kubajzz's toolbox - scripts in the first post by <<Kubajzz>>

F8ce version .8 parametric plugin - "f8cev8" script by <<BOB_I_Ts>>

Anim8or V 1.0 build 1.01.1318 Manual

I'm working on an XSI export script modifying Raxx's BZII X exporter script - scripts by <<Ippena>> and <<Raxx>>

RIBRobin Toolkit - "RIBRobin Toolkit v.0.2a" by <<Raxx>>

MorphIt v1.0 - "Morphit v0.1d" programm by <<Raxx>>

.STL Export Script - "STL Export" script by <<Raxx>>

Advanced Modeling Tool-Set (now 31+ last Update: 26.12.2013) - "PolytError" scripts by <<polyGon_tError>>

Mesh2A8s Export Script - "Mesh2A8s" script by <<Raxx>>, " 09cr8plugin" by <<Llyr Carter>>

Scripts Request: Edge to point or Faces to point! - "Sel_EdgesorFaces_to_Sel_Points_v01" script by <<NickE>>

LIA - Links In Anim8or - "LIA v1.1" programm by <<Raxx>>

XSI Export Script - LIA - "XSI_export_LIA" script by <<Raxx>>

AOBake - "AOBake" script by <<Raxx>>

Involute Gear Script - "Involute_Gears" script by <<NickE>>

Allthread Script - "Allthread" script by <<NickE>>

Advanced Spherize - "Spherize" script by <<Raxx>>

Select Larger/Smaller - "Select Larger Smaller" scripts by <<Raxx>>

ASL Editor [version 3.5 is out!] - "ASL Editor" programm by <<Kubajzz>>

Reload Textures plugin - "ReloadTextures" script by <<Kubajzz>>

MDL to an8 converter - "mdl2an8" script by <<Deepthought>>

BVH File Import into Anim8or - "BVH2Anim8or_20150703" programm by <<NickE>>

IconCreator for ASL - "IconCreator_v12" programm by <<Kubajzz>>

Mesh to Mesh Morphing Hack - "ShrinkWrap_Target", "To_Sphere" scripts by <<NickE>> and two files in the last post by <<B_twist>>

Icon Maker - Excel version - "Icon_maker" programm by <<B_twist>>

SCRIPT REQUEST! - 2D cell shading (AUTO GENERATOR) - "CelShadurr v0.3" script by <<Raxx>>

Animated Textures Using Controller Scripts - "Animated Textures with Controller Scripts - Resource" files by <<Raxx>>

Find tri's, quads etc in model - "mark_quads_1" script by <<NickE>>

Marching Cubes - Mesh of the Surface from a Collection of Points - "MarchingCubes_v0.4" script by <<NickE>>

Spiral shapes: cave, cone - "S_Tunnel", "Spiral_0 V2", "Spiral_Ec" scripts by <<B_twist>>

Metric Screws and Nuts Script - "Metric_Screws" script by <<NickE>>

PHUR v1.4 - "PHUR v1.4" script by <<Raxx>>

f8ce v0.9 plugin - "f8ce9tester" script by <<BOB_I_Ts>>

Boolean operation problem! - "0000_booleanoperations_7c_nf_opt" script by <<NickE>>

UVTools Preview - "UVTools v1.1" scripts by <<Raxx>>

There's no hope! I'm gonna nail it !!! 8)

License

The License can be found under the "About" button found on the top menu bar.

Terms of Use

1. OWNERSHIP:

Anim8or® is a copyrighted work owned by R. Steven Glanville.

The name Anim8or is a registered trademark of R. Steven Glanville

2. LICENSE:

You are granted a license to use Anim8or for personal, educational and commercial purposes. Anything that you make with it, you own. This includes images, models, animation projects, movies, etc. Of course if you use others work as part of your own you need their permission as well, but Anim8or imposes no restrictions on what you can do with your work. There is no fee for this license.

3. COPIES:

You may make copies of the Anim8or executable file and manual and distribute them.

You may post them on a server and make copies on CDs or other digital media and give them to your friends, students, co-workers and neighbors. You may NOT alter the anim8or.exe executable file in any way or make any claims that Anim8or is your property.

4. WARRANTY:

USE ANIM8OR AT YOUR OWN RISK. THERE IS NO EXPRESS OR IMPLIED
WARRANTY AS TO ITS CAPABILITIES OR FITNESS FOR ANY PURPOSE.

Anim8or is just a program that I write in my spare time and give away at no cost.

I can't promise anything but that I will try to fix any problems that my time allows.

BY USING THIS SOFTWARE YOU AGREE TO THESE TERMS.

I know your mother has told you this a hundred times but I'll say it again:

SAVE YOUR WORK OFTEN!

I hope you find Anim8or fun and useful.

Steve

Copyright 2017 R. Steven Glanville

About the Author

Hello!

My name is Steve Glanville. I work as a software engineer for NVIDIA, where I help write device drivers for OpenGL. In a previous life I was a compiler writer, and founder of Silicon Valley Software. But I couldn't stay away from 3D graphics, particularly character animation. So I figured out what it was all about by reading some books, going to SIGGRAPH, writing little graphics thingies, and taking a few Animation classes at nearby DeAnza Junior College. There I made friends worked with some really talented artists. It wasn't hard to see that I was better at writing code than being creative. So that's what I did.

Skip forward a couple of years and you have Anim8or. It's a work-in-progress, and I suppose it always will be. Please let me know of any problems, or new things that you might like to see added. I hope that Anim8or can be of some value to you in pursuing your life's dreams.

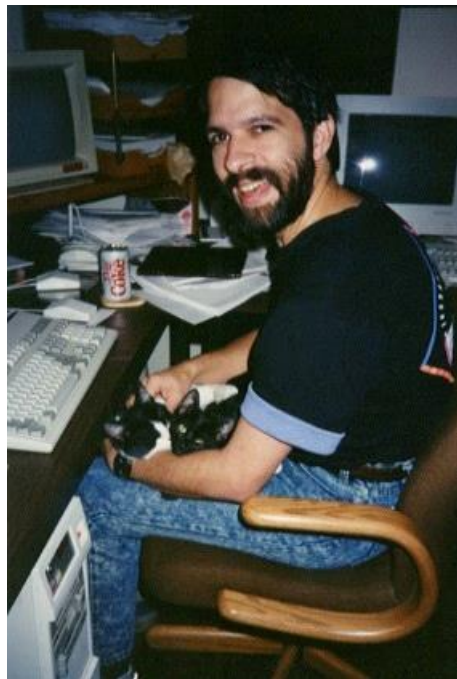
Enjoy!

Steve

R. Steven Glanville

rsteven@anim8or.com

website:www.anim8or.com



Hard at work with my two assistants,
Winchester and Colby

This page was last updated on November 10, 2013. Copyright 2013 R. Steven Glanville

Anim8or V 1.0 build 1.01.1318 Manual



The character is based on a cartoon figure my friends and I used to draw in junior high. You can tell why we didn't become artists by looking at it, but we had fun.



Anim8or® is a registered trademark of R. Steven Glanville.
Copyright 2018 R. Steven Glanville